SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Faculty of Mechanical Engineering

Reg. No.: SjF-5226-87787

BoBShield: a miniature ”ball on beam” experiment

Master thesis

2021 **Bc. Mgr. Anna Vargová**

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Faculty of Mechanical Engineering

Reg. No.: SjF-5226-87787

BoBShield: a miniature ”ball on beam” experiment

Master thesis

Study programme: Automation and Informatics of Machines and Processes
Study field: Cybernetics
Training workplace: Institute of automation, measurement and applied informatics
Thesis supervisor: prof. Ing. Gergely Takács, PhD.
Consultant: Ing. Erik Mikuláš

Bratislava 2021 **Bc. Mgr. Anna Vargová**

Slovak University of Technology in Bratislava
Institute of automation, measurement and applied informatics

Faculty of Mechanical Engineering
Academic year: 2020/2021
Reg. No.: SjF-5226-87787

MASTER THESIS TOPIC

Student: **Bc. Mgr. Anna Vargová**
Student's ID: 87787
Study programme: Automation and Informatics of Machines and Processes
Study field: Cybernetics
Thesis supervisor: prof. Ing. Gergely Takács, PhD.
Consultant: Ing. Erik Mikuláš
Workplace: ÚAMAI SjF STU v Bratislave

Topic: **BoBShield: a miniature "ball on beam" experiment**

Language of thesis: English

Specification of Assignment:

The student shall design and manufacture a miniaturized experimental module to control the position of the ball on a beam or inside a tube. The device will be used to teach technical disciplines in the general field of automation such as mechatronics, control theory, system identification and signal processing. The device will be compatible to the electronic layout of the Arduino R3 microcontroller prototyping board. The student will create a programming interface in C / C ++ for the Arduino IDE environment, for Simulink, or other programming languages and development environments. Another task is a mathematical-physical analysis of the dynamic process and subsequent experimental identification of the model. The student will also create various didactic instructional examples for feedback control of this system: PID control, linear-quadratic control (LQ), predictive control (model predictive control, MPC), or other control methods.

As part of the diploma thesis, the student must

– design the experimental equipment, select suitable electronic and mechanical components, design electrical connections and printed circuit boards, design a mechanical mounting, manufacture and test the functionality of the designed module;
- write an application programming interface (API) to control the device for various programming languages and development environments (C / C ++ for Arduino IDE, Simulink, or others);
- describe the underlying dynamic phenomenon by mathematical-physical analysis, design a suitable test signal and identify unknown model parameters based on the measured results;
- create didactic tasks of feedback control using PID, LQ and MPC control methods for each created environment.
- use Git and GitHub for software version management and integration into the parent library.

Length of thesis: 50-70 s.

Assignment procedure from: 15. 02. 2021

Date of thesis submission: 28. 05. 2021

Bc. Mgr. Anna Vargová
Student

prof. Ing. Cyril Belavý, CSc.
Head of department

prof. Ing. Cyril Belavý, CSc.
Study programme supervisor

Declaration on word of honour

I hereby declare that I am the sole author of this master's thesis and that I have not used other sources than those listed in the bibliography and identified as references.

Bratislava, 28. May 2021

.....
Signature

I would like to thank *To All the Boys Helping Me Before*: the guys from "Študovňa D205" for moral support; for more professional help to Bc. Martin Staroň and Ing. Matej Šimovec; my sincere gratitude belongs to my advisor prof. Ing. Gergely Takács, PhD. for his patience, enthusiasm and guidance along with Ing. Erik Mikuláš, whom I owe so many beers, he could open a pub.
I also owe some alcohol to Kata, who was ready to read this thesis, even she is not interested in automation at all.
And last, but not least I have to thank to my parents, for supporting my life long learning.
Bratislava, 24 May 2021 Bc. Mgr. Anna Vargová

Názov práce: BoBShield: miniaturizovaný experiment „guľôčka na tyči"
Kľúčové slová: Arduino, AutomationShield, experiment, identifikácia, riadenie, „ball and beam"
Abstrakt: Cieľom tejto práce je predstaviť čitateľovi zariadenie BoBShield, ktoré je jedným zo zariadení patriacich v rámci iniciatívy AutomationShield. Jedná sa o malú dosku plošných spojov, tzv. shield, obsahujúce periférie reprezentujúce experiment „guľôčka na tyči". Je to open-source zariadenie, kompatibilné s vývojovými doskami Arduino. Tak ako aj ostatné zariadenia rodiny AutomarionShield, aj BoBShield je určené na využívanie primárne pri výučbe teórie riadenia a výskum. V rámci tejto práce je opísaný samotný hardvér a dostupné rozhrania pre programovanie aplikácií v prostredí Arduinoi IDE a Simulink, s príkladmi použitia, sústredenými hlavne na riadenie. Okrem toho, práca obsahuje aj popis postupu pri modelovaní a identifikácií systemú, reprezentovaného týmto hardvérom.

Title: BoBShield: a miniature "ball on beam" experiment
Keywords: Arduino, AutomationShield, experiment, identification, control
Abstract: The aim of thesis thesis is focused on the introduction of the BoBShield device, which is one of the deivices belonging to the AutomationShield initiative. It is a small PCB, with peripherals representing the "ball on beam" experiment. The BoBShield is an open-source harware and software compatible with the Arduino boards. As the devices from the AutomationShield family, the BoBShield was created primarily for educational purposes, for teaching control theory. Within this work the available APIs are introduced for Arduino IDE and Simulink, along with examples of use, focused on control. Besides, the thesis contains a description of system modeling and identification for the experiment represented by the hardware.

Contents

List of abbreviations

API	application programmer's interface.
APRBS	amplitude modulated pseudo-random binary sequence.
FFC	flexible flat cables.
GA	genetic algorithm.
I2C	inter-integrated circuit.
IDE	integrated development environment.
ISR	interrupt service routine.
LQ, LQR	linear quadratic regulator.
MCU	microcontroller unit.
MIMO	multiple-input multiple-output.
ODE	ordinary differential equation.
PCB	printed circuit board.
PID	proportional-integral-derivative (controller).
PSD	proportional-sum-derivative (controller).
PWM	pulse width modulation.
SCL	synchronous clock.
SDA	synchronous data.
SISO	single-input single-output.
TOF	time of light.
USB	universal serial bus.

List of Figures

List of Tables

Introduction

The study of science and technology is mostly driven by the need to understand the principles and laws of various phenomena around us. In addition, successfully describing these phenomena can help us to properly manage, or even utilize some of them. Especially, when talking about automation we often encounter terms such as "modeling", which is basically about searching models that describe a particular phenomenon or "control" lying in finding the best method, in some predefined manner, to manage a phenomenon.

It is therefore natural that many devices were created to help people – mostly students and researchers – understand these phenomena. Thanks to these devices we can observe, investigate, or even improve the models describing a particular phenomenon represented by them.

Amongst the wide range of experiments which can be used to study various phenomena, there are some that are often used in control theory teaching and research. The ball on beam experiment, which we will discuss in this work, is one of them. Thus, it can be expected that there is a lot of literature dealing with this topic either on a more theoretical level, focused primarily on the modeling and simulations or publications in which some hardware is used and control algorithms are applied to the system briefly introduced in the Chap. 1. These devices often combine high standards for the accuracy of measurement and implementation, as their primary purpose is a professional analysis of the phenomenon and models usable to interact with this phenomenon. Taking into consideration the amount of available articles on the topic, one may wonder: why write another? Well, even if these sources approach the ball on beam system from many different angles, even from ones discussed in this paper, there are some aspects of the device introduced in this thesis, which make it unique and thus worth to write about. Likewise, while there are many purchasable devices representing the ball on beam system, the BoBShield can contribute to the wide range of the offer. Firstly, the discussed hardware is a part of a bigger project, the AutomationShield (described in more detail in Chap. 2), so the benefit of this particular device should be considered not just on its own, but as a contribution to a larger whole. On top of that, as all the devices from the AutomationShield, the created BoBShield has several advantages too:

- it is an open source hardware and software, so anyone can build, use and improve it;
- it utilizes affordable and accessible components, making the creation of such device really easy and cost-efficient;

- due to its dimensions, it is a compact and portable device;
- it can be used for didactic reasons, either by making students able to learn control theory via practicing on real hardware, or giving the opportunity to learn from own failures while building.

Besides, if the author of this work may express her personal opinion, the ball on beam experiment is one of the visually most appealing ones from the all experiments implemented in the AutomationShield project.

Discussing the implementation of the experiment, this thesis introduces the actually available hardware design of the BoBShield device, with a brief overview of its history in the Chap. 3, guides through the structure and creation of application programming interfaces in Arduino IDE and Simulink for the device in Chap. 4. In the Chap. 5 provides a simple explanation of the model possibly describing the system, along with the process of identification and estimation of parameters. After that (in Chap. 6) shows some examples of usage, applying PID control and LQ regulation to the system.

1 Literature Overview

As mentioned before, the ball on beam experiment is a very well documented experiment, so in this chapter some works dealing with this system will be introduced. The popularity of the system is originated in its complexity, in terms of mathematical modeling, as it is a non-linear and unstable system, not to mention how appealing the control of a hardware representing such a system can be. Firstly, papers focusing on the mathematical model of it will be shown. However, it is hard or nearly impossible to strictly divide the theoretical part of the problem from some hardware configuration, as the modifications of the models describing the system does not come just from a different approach but also from the differencies between each hardware configuration. Secondly, works, which are focust more on hardware design and on the building of a device represeting the system will be discussed. At this part some interesting implementations will be listed. And last, but definitely not least, works in which control algorithms are applied to the system will be discussed.

Amongst all documentation on mathematical modeling of the ball on beam system we have to mention the website Control Tutorials for MATLAB and Simulink [20]. As the name suggests, the website provides tutorials for the implementation various control theory experiments in MATLAB and Simulink, amongst others the ball on beam experiment as well. The importance of this source lies in the very professional and understandable explanation of the system and therefore the model too, making it a very useful starting point at a journey to understanding the ball on beam system. As there is a wide range of publications, the selection of other works to be introduced here is more of a cherry picking. An interesting conference paper was introduced in 2014 by Bolívar-Vincenty and Beauchamp-Báez [4], giving a proof of how the Euler-Lagrange methods and the Newtonian mechanics lead to the same model, while describing the importance of several acceleration terms especially for non-linear simulations. There is of course a plethora of research articles introducing applications of control algorithms on the system, which provide the mathematical model as well [21]. Based on the popularity of the system in control theory teaching and research, there are many commercial vendors offering a ready-to-use hardware such as Quanser BB01 Ball and Beam 1, AMIRA ball and beam Ball and Beam Control System Trainer Kit by ACROME, TecQuipment CE106 Ball and Beam Apparatus and others. The biggest advantage of these hardwares comes from the fact that they are standardized and tested, therefore they already have eliminated the most of bugs that can appear while designing a new hardware. However, purchasing such a device will not just harshly affect ones bank account, it also takes away the fun part of building a DIY device. Fortunatelly, building such a hardware does not

Figure 1.1: Quanser BB01 Ball and Beam [13]

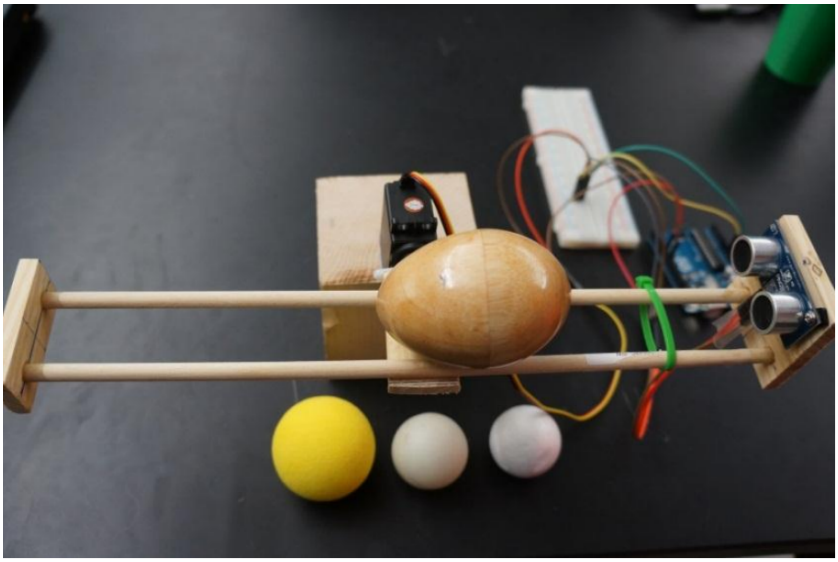


Figure 1.2: The setup of experiments with different balls [10]

require any special parts and there are people able and willing to do so. These people, whether researchers or students, often get creative, e.g. Gao and collective [10] not just built a ball and beam experimental device, they also tried to map the system dynamics with different sizes of ball and even replacing the ball with other geometric shapes as shown in Fig. 1. Other's creativity manifested in using untraditional and avaliable materials, such as lego building blocks 1 or cardboard 1. Despite these extravagant and maybe cheap-looking designs, there are examples of very elegantly constructed ball on beam devices as well 1. Note that many of these home-made ball on beam devices are based on Arduino too, thanks to its easy usage and availability.

Many of the works on the ball on beam topic are dealing directly with the design and tuning of a controller for the system, either using some hardware or only simulations. As expected, the most common control algorithm used in these papers is the classical or adaptive PID and LQ regulator. Some works try to compare various control algorithms [7]. Others apply more exciting algorithms, such as neuro-fuzzy [6], model predictive [11], sliding mode [15], fractional order control [5] and others.

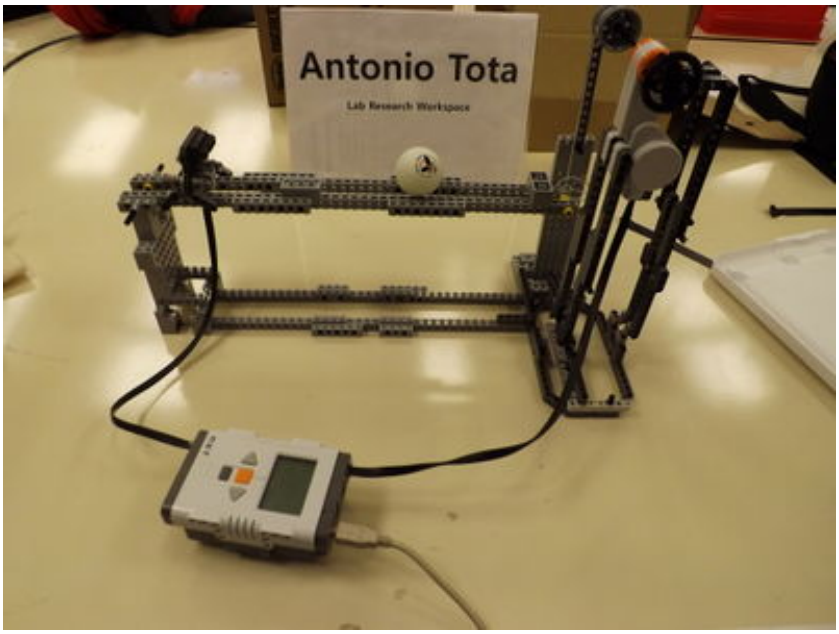


Figure 1.3: Ball on beam device made of lego building blocks [2].

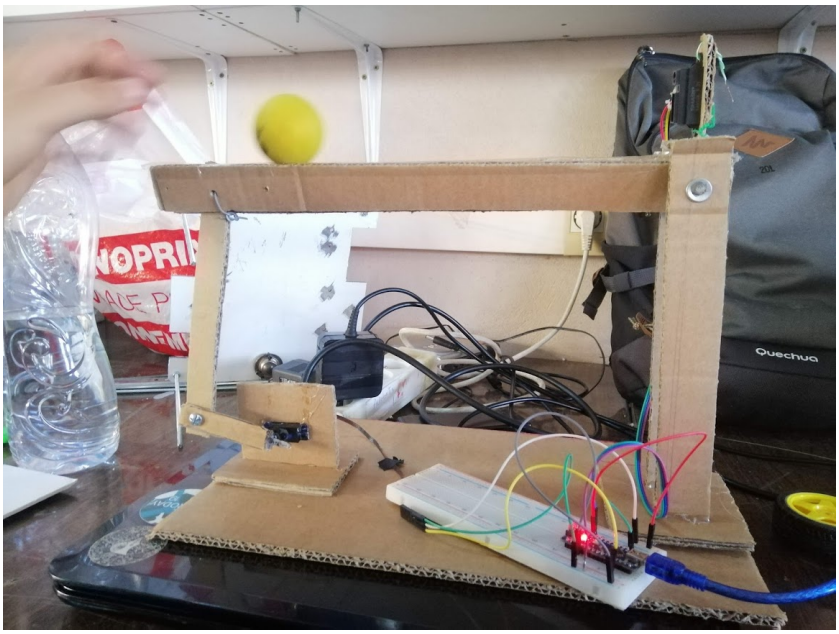


Figure 1.4: Ball on beam device made of cardboard [3].

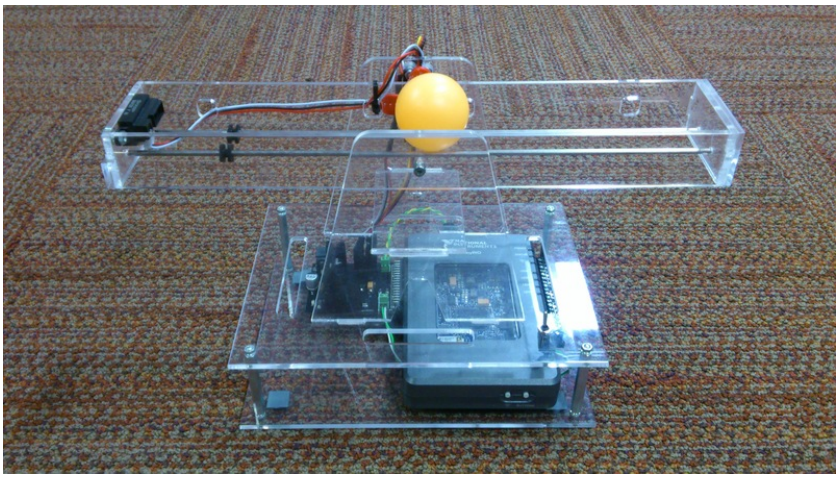


Figure 1.5: myBall& Beam - a student project [1].

2 BoBShield Origins

2.1 AutomationShield

As mentioned in Chap. , the BoBshield miniature experiment has been developed as a part of the AutomationShield open-source hardware and software initiative introduced in this section in a nutshell.

The main goal of the AutomationShield project is creating affordable tools for control engineering and mechatronics education including hardware design, application programmer's interface (API) and examples for students, educators and researchers. The core of the project are reference designs of extension modules for the popular Arduino microcontroller prototyping boards, which are implementing feedback control experiments to teach control systems engineering and mechatronics. These hardware extensions — known in the Arduino world as shields — are then in essence experimental systems on a single printed circuit board (PCB).

Currently the reference designs of these shields are available [16]:

- **MagnetoShield** - a magnetic levitation feedback experiment;
- **FloatShield** - a floating ball feedback experiment;
- **HeatShield** - a thermal system feedback experiment;
- **MotoShield** - motor speed and position feedback experiment;
- **OptoShield** - a low-cost optical feedback experiment;
- **LinkShield** - a rotational link feedback experiment.

BoBshield is one of the new types of shields, which is currently classified as beta version within the project, because it still has som flaws and is being tested. However, some parts of the project have already been published mostly as conference papers, and bachelor's or master's theses. Even the discussed BoBshield has been introduced on the EDUCON2021 - IEEE Global Engineering Education Conference. The author of this thesis had the oppportunity to contributed to this work [9], which is added in the appendix A (in print).

2.2 The Concept of BoBShield

Surpisingly, the name BoBShield does not refer to the actor[1], but it follows the usual AutomationShield naming conventions. Just like in the case of any other device that is part of the initiative, the second part of the name BobShield refers to the specific hardware design mentioned above: the shield. However, unlike others, for this particular shield the first part of the name does not refer to a physical phenomenon, but to the abbreviation of the well-known "Ball on Beam" system.

In spite of the many different designs of available devices and even theoretical configurations of the discussed experiment, it can assumed that all these implementations have some things in common, for example:

- there is a surface with a dominant dimension, so as a beam, guiding rails or a tube,
- there is always a sliding or rolling object present, usually a ball,
- and the goal of the feedback control system is to adjust the inclination such that a ball tracks a pre-set reference trajectory,
- so a suitable actuator, mostly an electric motor, must be included too.

On the other hand, depending on a particular configuration, devices for this experiment can differ in:

- the type of actuator – even the actuators are usually electric motors, the system input can be different, e.g. the beam angle or the moment,
- how and where the support surface is mounted to the actuator
- and last, but not least the sensors used to measure the ball position or even other units.

While these differences do not change the essence of the experiment, they can cause some changes in system identification and modeling, which we will discuss laterm int the Chap. 3,

In order to keep the shield compact and to have a design as simple as possible, the original creators opted for a design with the most availiable and user-friendly parts, with a servo motor, time-of-flight (TOF) sensor and a micro RC servo motor turning directly the support surface in its middle. This hardware design is described more in detail in the next chapter.

[1]Robert "Bob" Shield (1917-1996) was an actor, known for The Judge (1986), Hell's Half Acre (1954) and Science Fiction Theatre (1955) [12].

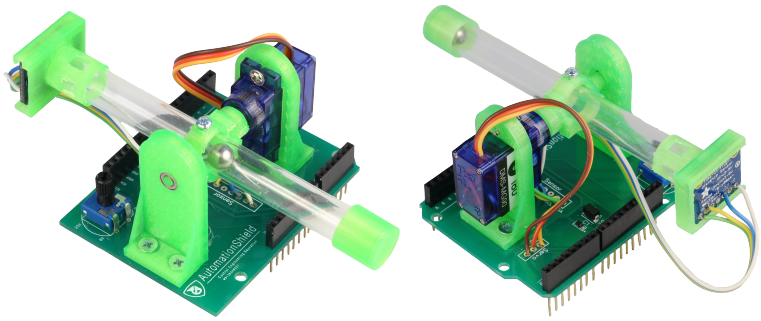
3 Hardware Design

Although each shield is considered a prototype, improvements of the software are made continuously and the hardware is being updated every time a sufficient number of improvements arise. In this chapter, the hardware design of both currently available versions are described. Despite the fact that in this work primarily the second version of the hardware - the BoBShield R2 - is discussed, firstly the R1 version should be introduced.

3.1 BoBShield R1

The BoBShield R1 (Fig. 3.1) was developed by a group of students as a part of a class project for course of Microcomputers and Microprocessor Technology. As all other shields from the AutomationShield initiative, the BoBShield is built on a printed circuit board copying the outline of the Arduino Uno - labelled as (a) in Fig. 3.1. The used two-layer FR4 board (Fig. 3.6) with 1.6 mm thickness and standard manufacturing tolerances fits into the customary 100 mm $\times$ 100 mm size limit, allowing one to utilize the several small batch PCB manufacturers offering extremely competitive pricing. With these specifications and design the BoBShield may be connected and mechanically fixed to any R3 pin layout microcontroller prototyping board. At first sight one will notice a 100 mm long transparent plexiglas (b) connected to a servomotor (c), which is mounted on the shield. The tube has an inner diameter of 10 mm and holds the 8 mm steel ball (g) that moves freely along its longitudinal axis, if inclined. One end of the tube is closed by a 3D printed cap (d), while the other end is terminated by another 3D printed part designed to hold the distance measurement unit (e). The tube itself is fixed at its middle by a 3D printed circular clamp (f) that permits the inclination of the tube along its major axis. The clamp is held in place at both sides by L-shaped 3D printed brackets (h1) and (h2), one containing a miniature ball bearing to facilitate smooth movement and the other fixed to the clamp through the shaft of the actuator [9].

As the actuator of the system, serves a standard micro RC Servo motor with analog feedback. This manipulates the inclination of the tube and through this the position of the ball. To make the whole device simple and the total rotation range of the motor utilized in this application is only $\pm 30°$, thus any identically sized servo actuator will be suitable. Finally, there is a potentiometer with a small plastic shaft that may be used for any user programmable role; but the main idea is to provide means to manually input the desired reference signal. The absolute distance of the ball from the reference located at one end of the tube is measured



(a) Front view. (b) Back view.

Figure 3.1: Isometric view of the assembled BOBShield R1.

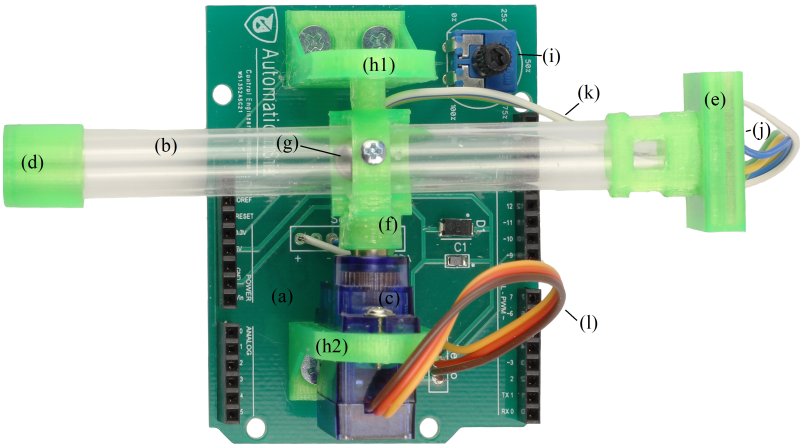


Figure 3.2: Top view of the assembled BOBShield R1 showing component marking.

by the ST Microelectronics VL6180X time-of-flight (TOF) sensor [9].

As there are no electric parts that need very sophisticated wiring, the electric circuit is kept very simple, as shown in Fig. 3.1. The motor is connected to the D9 pin of the standard Arduino R3 layout, since this equivalent MCU pin handles interrupts and timing on the Uno as well as other prototyping devices. The servo motor is directly powered from the 5V supply of the Arduino board without the need of an external wall-plug adapter. A capacitor was added parallel to the motor to smooth out possible transients affecting the board supply and a diode to prevent possible back-electromagnetic force damaging the circuitry.

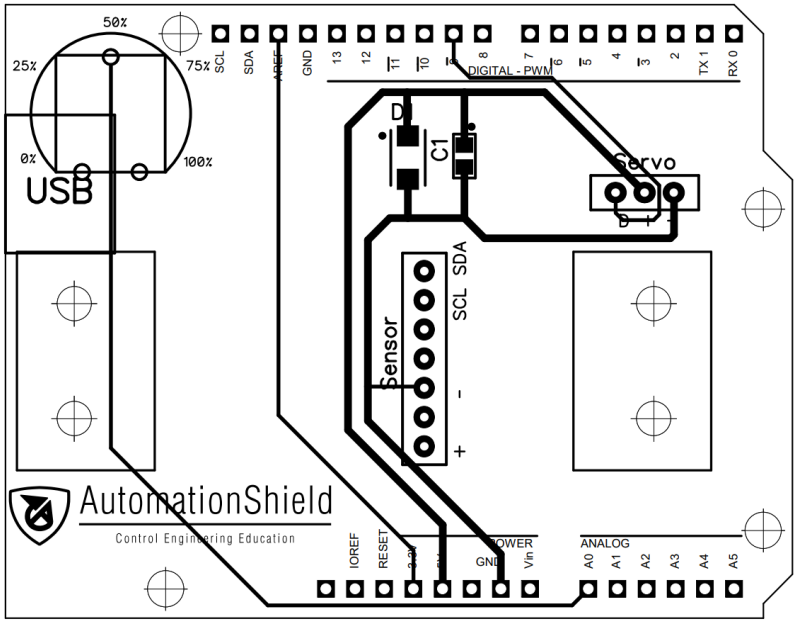
3.2 BoBShield R2

As mentioned before, new hardware releases occure only after a certain time, with a reasonable amount of changes. In case of BoBShield device, series of minor flaws led to the developement of a new hardware. These flaws were mostly related to the arrangement of some parts and the design of some 3D printed elements, but with the original purposes of the parts preserved. It can be nicely seen in Fig. 3.2, where the differently looking parts are labeled with the same letter, as in Fig. 3.1.

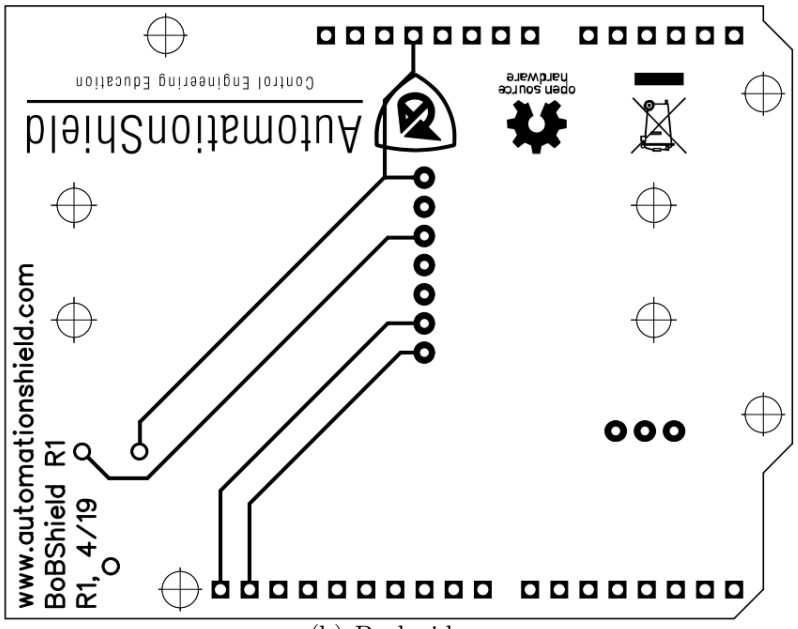
In the original layout, it was difficult to manually set the reference using the potentiometer as it was too close to the tube path. Due to this, it was necessary to move the plexiglass tube, to make the space between the potenciometer and the tube wider. As a part of the new release, the clumsy cables leading from the sensor to the board were also replaced. The R2 version uses flexible flat cables (FFC), insted of classic ribbon cables, led along the tube. Thanks to this change, the new hardware looks more elegant and professional. In addition, the selected new cable in such configuration is more resistant to failures. The new PCB can be seen in the Fig. 3.2.

With the cable replacement came some additional changes, especially in the connection of the cable to the board and to the sensor. On the board itself, this change is reflected only by adding a connector suitable for the new cable. On the other hand, connecting a cable to a sensor that is already mounted on a breakout required a more creative approach. Therefore a breakout to the breakout was created. The purpose of this simple breakout is to bridge the difference between the spacing of the breakout contacts with the sensor and the cable, more precisely the connector. This breakout is shown in Fig. 3.2. Another future possible improvement could be to design another breakout, which would integrate the sensor with the relevant electronics and at the same time the connector for the FFC cable on one tiny board.

Within the new developement, it was also necessary to solve the problem with the precision of the plexitubes inner diameter as smaller plastic tubes often do not have guaranteed inner dimensions a compromisewas chosen. For the release of the R2 a thicker tube, commonly used for PC cooling systems was used. Even the inner dimensions of these tubes are not guaranteed, they provide much higher precission, just enough to not cause visible stops when the ball is sliding along its way. This change of the tube resulted also in the change of the ball for a bigger one. Note that for the proper flow of th experiment this shold not be necessary, but it is more



(a) Front side.



(b) Back side.

Figure 3.3: The printed circuit board of the BoBShield R1.

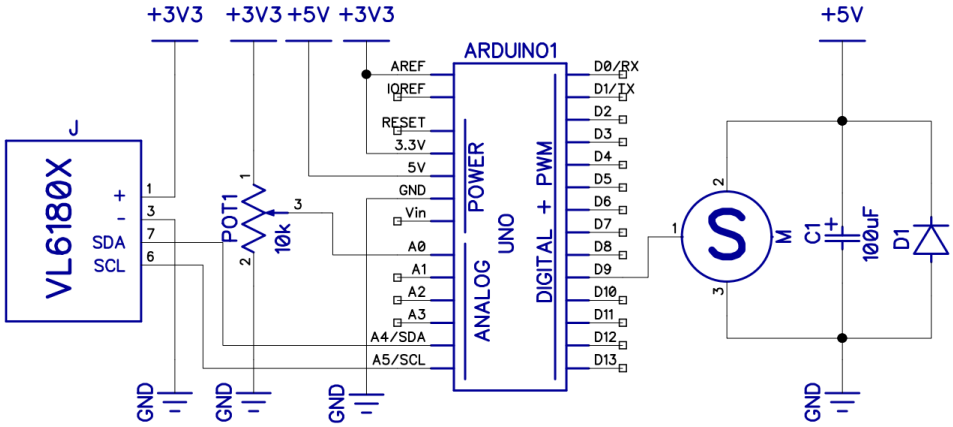


Figure 3.4: Electronic schematics of the BoBShield R1.

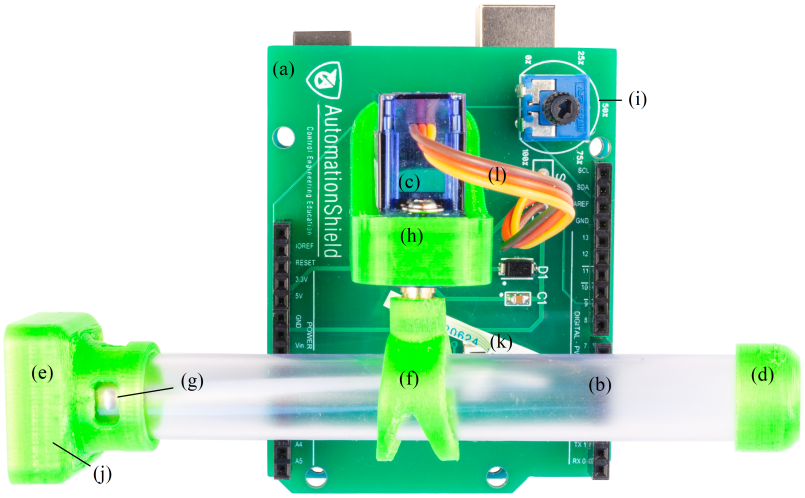
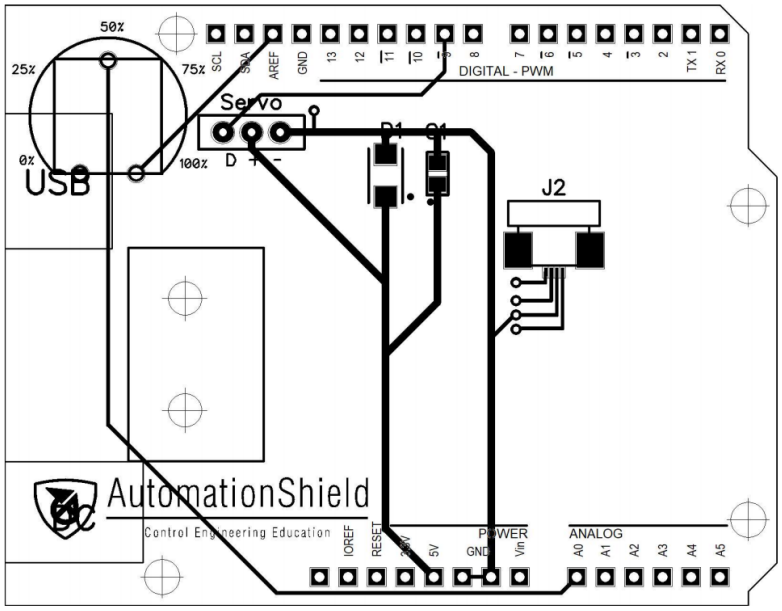
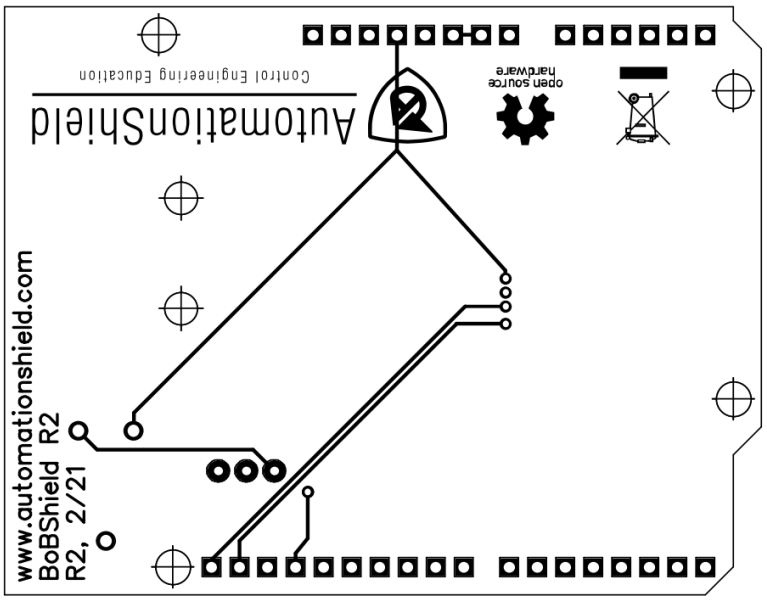


Figure 3.5: Top view of the assembled device showing component R2.



(a) Front side.



(b) Back side.

Figure 3.6: The printed circuit board of the BoBShield R2.

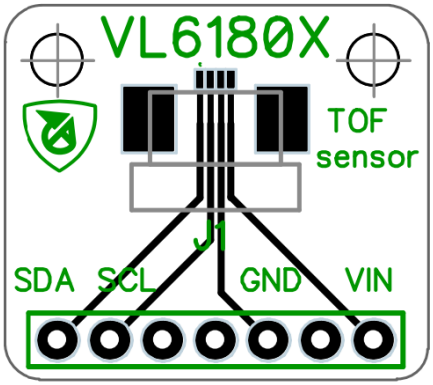
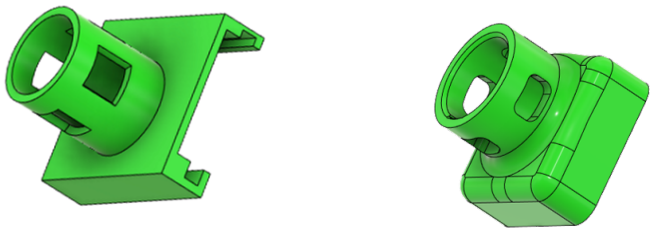


Figure 3.7: The printed circuit board of the breakout.



(a) R1. (b) R2.

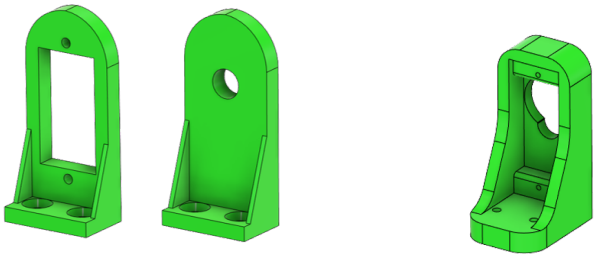
Figure 3.8: The sensor case for BoBShield R1 and R2.

convenient, as with the right tube cross-section to ball diameter ratio the experiment naturally uses the air pressure in the tube to slow down the ball. This effect is not directly considered in any further deduction, but is certainly incorporated in the constants used in examples.

Other changes are related only to the design of the 3D printed parts. The sensor case was changed (Fig. 3.8) to allow securing of both the sensor and the bridging breakout to it with the help of 2 screws. Also the new sensor case allows the FPC cable to be routed parallely with the tube. A new part (Fig. 3.2) was added to stop the ball securely before reaching the sensor, preventing it from any mechanical demage potentially caused by the ball. Also the attachement of the tube to the motor was changed. The two L-shaped parts and the original circular clamp were replaced with only one L-shaped part (Fig. 3.10) used to hold the servo motor and with a new clamp (Fig. 3.11) allowing to mount the tube directly to the axis of the motor using a screw. More over, the new clamp fixes the tube with a screw off the cross-section of the tube decreasing the tube deformation which can affect the path of the ball. Lastly, the blinding at the end of the tube was changed too, because of the bigger diamater and to match the new design (Fig. 3.12).

The final layout of the BoBShield R2 is shown in Fig. 3.2, utilizing the components listed in Tab. 3.1

Figure 3.9: Top view of the assembled device showing component marking.



(a) R1. (b) R2.

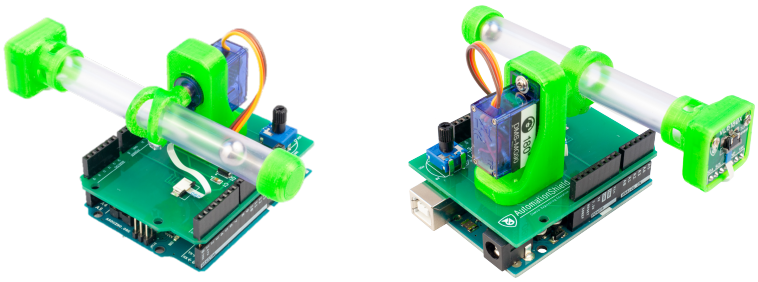
Figure 3.10: The sensor motor stand for BoBShield R1 and R2.

(a) R1. (b) R2.

Figure 3.11: The tube holder for BoBShield R1 and R2.

(a) R1. (b) R2.

Figure 3.12: The blinding at the end of the tube for BoBShield R1 and R2.



(a) Front view. (b) Back view.

Figure 3.13: Isometric view of the assembled BOBShield R2.

Table 3.1: Component list - BoBShield R2.

Name	**Part no.**	**PCB**	**Mark**	**Pcs.**
PCB - breakout	2-layer, FR4, 1.0 mm thick	-	-	1
Connector	FFC Board Connector, ZIF, 0.5 mm, 4 Contacts	J2	-	1
Cable	Ribbon FFC CablePitch 0.5 mm, 4P wire, cca.10CM	-	-	1
PCB	2-layer, FR4, 1.6 mm thick	-	(a)	1
3D print	18 g, $\phi = 1.75$ mm PETG filament, bright green	-	-	1
Trimmer	10 k\Ω, 250 mW	POT1	(i)	1
Screws	DIN 7981F 2.9 × 6.5	-	-	3
Diode	generic diode, DO214AC	D1	-	1
Screws	DIN 912 M2,5 × 6	-	-	2
Screws	DIN 7981C 2,2 × 4,5	-	-	2
Header	10×1, female, stackable, 0.1“ pitch	-	-	1
Pot shaft	For “POT1”	-	-	1
Header	8×1, female, stackable, 0.1“ pitch	-	-	2
Header	6×1, female, stackable, 0.1“ pitch	-	(b)	1
Tube	transparent, plexiglass, L=100 mm, 12/10 mm	-	1	1
Sensor	VL53L1X breakout, time of flight ranging SNSR	J	(j)	1
Motor	Metal geared 9 g, 5 V micro servo	M	(c)	1
Capacitor	100 uF, 6.3V, 20, 0805, tantalum	C1	-	1
Ball	Steel ball, diameter 8 mm	-	(g)	1
Connector	FFC Board Connector, ZIF, 0.5 mm, 4 Contacts,	J2	-	1
Cable	Ribbon cable, 4-7 wires, cca. 0.2 m (e.g. EL-2468)	-	(l)	1

As the main goal of the creation the BoBShield device is its further usage in control process teaching, it is highly important to make as precise measurements as possible. Due to that, by the release of R2 some other ways were considered, in order to the measuring of the ball distance more accurate. The precision of the ball position measuring is mostly affected by the sensor itself and the ball-surface. Since the choice of the sensor was based on its availiability and easy usage, changing the sensor would not be effective. The other approach is to find the best ball-surface, considering different levels of reflectivity. Therefore, a simple experiment of measuring different surfaces was conducted, where surfaces with various reflectivity were placed in front of the sensor, at an equal distance of 100 mm. The results are shown in the Tab. 3.2, where $\overline{x}$ is the mean and σ is the standard deviation of measurements.

Table 3.2: Surface quality.

	High reflectivity	**Medium high reflectivity**	**Medium low reflectivity**	**Low reflecitivity**
$\overline{x}$	97.84	94.83	94.26	70.51
σ	2.68	1.55	1.25	1.42

After obtaining the results, the following assumptions were made:

- low reflectivity surfaces may result in a larger measurement error;
- even the average of measurements made on high reflective surface is the closest to the expected value, these measurements have the highest standard deviation;
- medium high and medium low reflectivity surfaces have similar properties.

Despite that these observations may suggest that it would be better to opt for some medium reflectivity surface, the availiability of the steel balls won over the efficient parameters. This may not seem as the best idea, but knowing how the measurements are affected by the surface may be helpful too, especially when estimating the ball position based on the measurements.

4 BoBShield API

One of the goals of the AutomationShield project is making the usage of every device possible in more than one programming environment. That means, libraries or so-called Application programming interfaces (API) are developed in programming languages as:

- Arduino IDE
- MATLAB
- Simulink
- Python.

Note that not all devices have fully developed API in all languages mentioned above. This can be due to some technical limitations making the implementation impossible in one or another language, or justdue to the fact, that the development a particular device has not gone so far yet. All versions of the API implemented in different languages contains basically the same methods and functions, only adjusted to the requirements of one specific environment.

4.1 Arduino IDE

Given that one of the reasons why the devices of the AutomationShield project are developed on boards compatible to Arduino is the simple programming of these microcontroller boards via the cross-platform application (for Windows, macOS, Linux) used to write and upload programs to Arduino compatible boards called Arduino Integrated Development Environment (IDE). As a natural result, application programming interface in Arduino IDE is available for all the devices from the AutomationShield family. With the release of the BoBShield R1, a first version of the API was released too. This application programming interface contained several methods allowing the user to easily manipulate with the hardware with a set of understandable commands with a structure respecting the AutomationShield standards and conventions. During further development some minor changes were applied and some technical issues and flaws were removed.

The BoBShield library is written in one single header file, which is not the standard approach in the AutomationShield family. Usually libraries for a particular device consist of both a header file and a .cpp file, but in our case, this was not possible regarding the implementation of the servo library, but this fact does not

limit the usage in any way. This means that this header file incorporates both the methods and the class definition, defining the private and public members of the BoBShield class:

```
class BOBClass {
  public:
    Adafruit_VL6180X sens =  Adafruit_VL6180X();
         //symbolic name for sensor library
    void initialize();                                    //
       sets up the sensor - required to run in setup!
    void  begin(void);
         //sets up sensor
    void calibration();
         //calibration
    float referenceRead();
                                  // Read reference pot in
        %
    float sensorReadPerc();
    float sensorRead();
    void actuatorWrite(float fdeg);

  private:
      float _referenceRead;
      float _referenceValue;
      uint8_t range;
      float _servoValue,pos,posCal, position,ballPos;
      int  posperc;
      float maxRange, minRange;
      byte calibrated = 0;
      float minCalibrated = 500;
                        // Actual minimum value
      float maxCalibrated = 10;
                                  // Actual maximum value
  };
```

As the BoBShield device uses the Adafruit VL1680X time of flight sensor, which communicates via i2c, and a servo motor, besides the usual `Arduino..h` and `AutomationShield.h` other libraries are included into the BoBShield header file such as:

- `AdafruitVL6180X.h` making the methods from the sensor API available,
- `Wire.h` makin an easy communication possibble via I2C,
- `Servo.h` for controlling the servo.

Naturally, to prevent multiple library inclusions, include guards are used. The current library consists of seven methods.

- `BOBShield.begin();` Is the void method, a method with no result value, serving to initialize the servo motor via assignment of the pulse-width modulation (PWM) pin of the board.

```
void BOBClass::begin() {
  myservo.attach(BOB_UPIN);      // set Servo pin
}
```

- Another initial method that needs to be included at the beginning of any script for using the BoBShield device is the `BOBShield.initialize();` method. With the help of this method the sensor object is created. Even though it is not common for the devices from the AutomtionShield family to have more than one initial method, it was beneficial for us to do so, allowing to initialize both the sensors separately. This made possible for the time of light sensor do some basic troubleshooting, to check if the sensor is connected properly. The script of this method then looks like this:

```
void BOBClass::initialize() {
        sens.begin();
         # if ECHO_TO_SERIAL
      Serial.println("Adafruit VL6180x test!");
  if (! sens.begin()) {
   Serial.println("Failed to find sensor");
    while (1);
  }
 Serial.println("Sensor found!");
    # endif
}
```

- `BOBShield.calibration();` method is used to calibrate the device in terms of finding the minimal and maximal measured distances from 100 measurements. This is necessery for proper usage of methods that use the ball distance as a percentual value of the whole reachable length. Note that the minimal and maximal values are preset, so not calling this method does not result in the disfunctionality of these methods, they just will not be accurate.

- Method `y=BOBShield.sensorRead();` of the BoBShield class is used to assign the measured ball distance to a variable in milimeters. This method is basically only the wrapping one of the functions provided directly from the sensor library into the BoBShield class as it can be seen below:

```
float BOBClass::sensorRead(){
        if (calibrated == 1)   {
                                    // if calibration
           function was already processed (calibrated
            flag ==1)
 pos = sens.readRange();
```

```
 ballPos  = pos - minCalibrated;                    //
    set actual position to position - calibrated
    minimum
}
   else if   (calibrated ==0) {
                              // if calibration
      function was not processed (calibrated flag
      ==0)
  pos = sens.readRange();
 ballPos  = pos - MIN_CALIBRATED_DEFAULT;          //
    set actual position to position - predefined
    value
}
  # if ECHO_TO_SERIAL
Serial.print("pos :"); Serial.print(pos);Serial.print
   (" ");
 Serial.print("ballPos :"); Serial.print(ballPos);
    Serial.print(" ");
    # endif
return ballPos;
}
```

- Similarily, the user can obtain the measured ball distance as a percentual value from the reachable length of the tube using the `y=sensorReadPerc();` method. While this method is a simple conversion value from the method `y=BOBShield.sensorRead();` using the measured or preset, minimal and maximal distances:

```
float BOBClass::sensorReadPerc(){
 range = sens.readRange();

    if (range < minCalibrated) {range = minimum;}
    else if (range > maxCalibrated) {range = maximum
       ;}


  posperc = map(range,minimum,maximum,0,100);
 return posperc;                //returns the ball
    distance in 0 - 100 \%
 }
```

- As all devices in the AutomationShield project, the BoBShield also has a potentiometer allowing to set the reference manually. The desired value set like this can be obtained as a percentual value for the program using the method `r=BOBShield.referenceRead();`, which is defined as:

```
float BOBClass::referenceRead(){
```

```
    _referenceRead = analogRead(BOB_RPIN);
    _referenceValue = AutomationShield.mapFloat(
       _referenceRead,0.00,1023.00,0.00,100.00);
  return _referenceValue;
}
```

- Finally, the angle u (limited from -30 to +30) is sent to the servo motor using the method `BOBShield.actuatorWrite(u);` scripted as follows:

```
void BOBClass::actuatorWrite(float fdeg){
        deg= (int) fdeg;
                                              // scale
           input float parameter into integer (servo
           works only with integers)
if (deg<-30) {
   // predefined boundary for servo range, to prevent
    hardware damage (in degrees)
  deg=-30;
}
else if (deg>30) {
  deg=30;
}
```

When coming to angles, it is natural that in some cases it is easier to use angles in degrees, but sometimes a user may need to use radians. Therefore two additional methods where created and added to the AutomationShield class, as:

- `u=deg2rad(float u);` to convert degrees to radians as:

```
float deg2rad(float u){
    u=u*(3.14/180.0);
    return u;
    }
```

- `u=rad2deg(u);` to convert radians to degrees as:

```
    float rad2deg(float u){
    u=u*(180.0/3.14);
    return u;
    }
```

4.2 MATLAB

Unlike the Arduino IDE, in the case of MATLAB, the written program is not compiled and executed directly on the Arduino board chip, but on the user's computer, and only individual commands are sent subsequently to the hardware. In some cases, this can cause problems resulting mainly from unsecured correct timing, in our case

the functionality of the API should not be compromised. When creating the versions of the application programming interface for any device from the AutomationShield family on more than one platform, there is always an effort to maintain the structure as similar as it is posibble in different languages. This applies for the BoBShield too, meaning that the MATLAB library also should contain methods like:

- Initialization
- Reading from the sensor to provide this functionality, we used the existing sensor library for applocation in the Arduino IDE, but it was necessary to "wrap" it in the MATLAB structure.
- Turn the servo Implements angle rotation notation, which is enabled within the default libraries available in MATLAB for Arduino.
- Reading from a potentiometer

Based on the specifics of the MATLAB environment, and on the fact that the code will basically run on the user's computer, in this case there would be no benefit from initializing both the sensor and the servo separately. Thus, the initializing method should handle the whole initializing process, including the creation of an arduino object in MATLAB, the pin assignment to the servo based on the actual hardware configuration and also the creation of the sensor object.

4.3 Simulink

Even though Simulink can not be treated as a separate language, as it is a part of the MATLAB environment, it should be mentioned as an other environment to create an API in. Simulink is a block diagram environment for Model-Based Design. In terms of programming an Arduino board, the main advantage of Simulink towards MATLAB lies in the fact that using Simulink the program is executed directly on the board, just like in case of the Arduino IDE. On top of that, the design of Simulink, as it uses graphical units, each representing an algorithmic block, connected with lines to mark the information flow through the model, making the presentation of an examined object (usually a system) visually more comprehensible. The term model in Simulink is primarily referring to the file containing these blocks and lines, not just the model of a system as we are used to it.

Any program to the Arduino board is created in Simulink as a Simulink model using either the available set of blocks, or creating custom ones. As a part of this work, a basic set of commands in a form of blocks was created. The library in this sense is a Simulink model containing all program blocks, which the library file will be created of. Note that a custom-made library can be searched in the Library Browser in Simulink only if the library file is saved to the MATLAB Search Path, or after running the `installMatlabAndSimulink` file included in the AutomationShield library.

In case of BoBShield device, the basic commands that should be implemented in Simulink are used to:

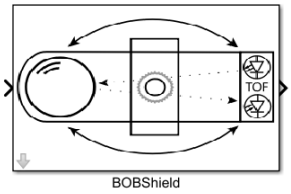


Figure 4.1: BoBShield block

- read the ball position from the sensor,
- write the input to the actuator,

and for setting the reference manually

- read the reference value from the potentiometer.

Besides, a separate block called BOBShield, Fig. 4.3, is needed as well, handling the system input, output and value settings. This block contains a subsystem consisting of: Matlab function for BoBShield block functionality, providing the calculations of the outputs to be passed and calibration. The inputs of the funcion that can be set directly from the dialogue box of the block are:

- sampling period,
- number of calibration samples
- output type option (whether the ball position should be passed from the block in mm or as a percentage of the reachable length of the tube),

Other input obtained from the connected hardware is the

- raw sensor reading.

and the last one that is passed from outside of the block is the

- input servo angle.

The outputs of the function for BoBShield block functionality are

- servo angle sent to be set on the hardware,
- and the ball position to be passed to the output of the block.

The subsystem of the BOBShield block is shown in Fig. 4.3 For a wide range of applications, only the use of the Bobshield block is needed, unless the user wants to control the BoBShield manually. For this purpose, the Reference Read block was created along with the Sensor Read block and Actuator Write block. The Sensor Read block (Fig. 4.3) is used to obtain the ball position. In this block the output type, Calibration values and the sampling period can be set. This block utilises a block called an S-function block, which enables the use of external C and C++ code, which in this case was necessary, as the VL6180X TOF is highly dependent on the

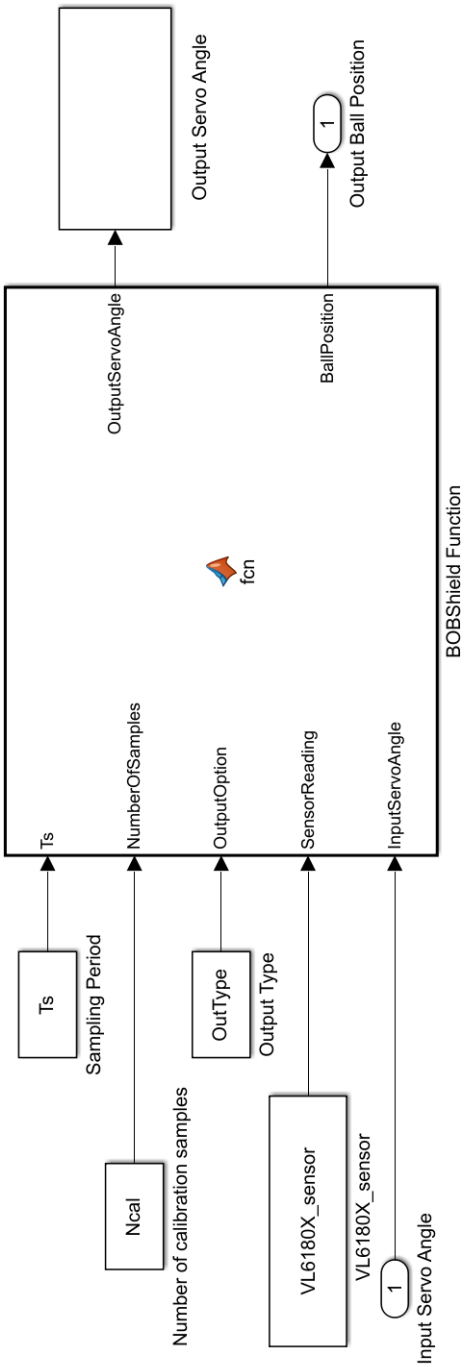


Figure 4.2: BoBShield block - Subsystem

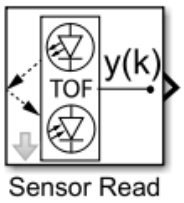


Figure 4.3: Sensor block

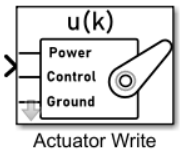


Figure 4.4: Actuator block

C++ library provided by the manufacturer. Note that this is the same S-function, that is used in the BOBShield Block.

The Actuator Write block shown in the Fig. 4.3 enables to the user to set the servo angle with the possibility to choose the input type, and set the maximal and minimal values of the angle. This block uses the Arduino Standard Servo Write block of the Simulink Support Package for Arduino Hardware add-on to pass the calculated value into the hardware.

The last block of the BOBShield library is the Reference Read block (Fig. 4.3) created in line with the AutomationShield project standards.

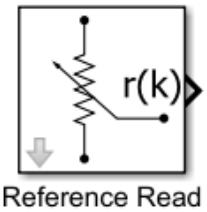


Figure 4.5: Reference block

5 Modeling and Identification

5.1 Modeling

The importance of having a mathematical model describing the behavior of the ball on beam system can be derived from the desire of being able to control this system with more advanced control techniques than e.g. a simple PID regulation. Fortunately, the ball on beam system is widely known, and used in many experiments and publications (c.f. [20], [4], citeVirseda2004), meaning that there is no need to reinvent the wheel and some of the well-known models of this system can be introduced below.

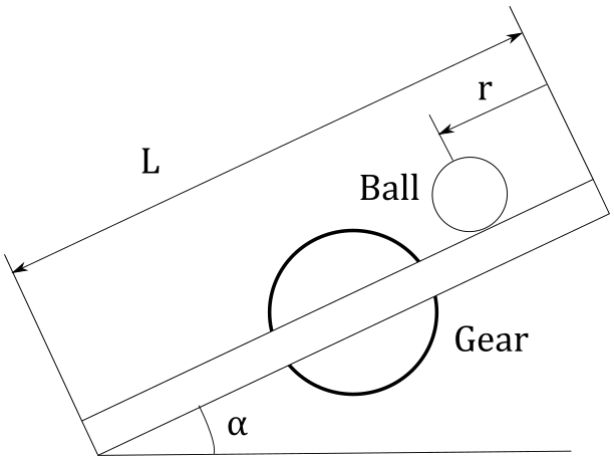


Figure 5.1: Ball dynamics

Firstly, based on the system arrangement shown in Fig. 5.1, let us assume that there is no friction, or in other words, the ball is sliding down the beam, instead of rolling. In this simplified model, the ball moves along the beam with acceleration, coming from the component of gravity that acts parallel to the beam. The equation of motion for the Ball is given by the following equation [14]:

$$\sum F = M * a = M * g * \sin(\alpha), \tag{5.1}$$

where M is the ball mass, a the acceleration, given by the product of g, the gravity and $\sin(\alpha)$, with the beam angle α. Substituting $\frac{d^2x}{dt^2}$ for acceleration a and manipulating gives:

$$\frac{d^2x}{dt^2} = g * \sin(\alpha), \tag{5.2}$$

where x is the ball position. This model can be linearized using the fact that for small angles $\sin(\alpha) \approx \alpha$ as

$$\frac{d^2x}{dt^2} = g\alpha, \tag{5.3}$$

or in state space form we can write:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ g \end{bmatrix} u \tag{5.4}$$

As in the hardware configuration of the BoBShield device corresponding the arrangement described in Fig. 5.1 the input to the system is directly the servo motor angle, the model given above needs no further transformation. There are two forces influencing the motion of the rolling ball, which are: F_{tx} : the force due to translational motion and $F_r x$: the force due to ball rotation [14]. While the force due to ball rotation can be derived from the torque T_r as follows:

$$T_r = F_{rx} R, \tag{5.5}$$

ref with R being the radius of the ball. The torque is equal to the ball's moment of inertia J multiplied by its angular acceleration $\frac{d^2\omega}{dt^2}$, which can be written as the second derivative of the andle x devided by R, giving:

$$T_r = F_{rx} R = \frac{J}{R}\ddot{x}. \tag{5.6}$$

Now by substituting for $J = \frac{2}{5}MR^2$ the force F_{rx} can be written as:

$$F_{rx} = \frac{2}{5} M \frac{d^2x}{dt^2}. \tag{5.7}$$

As the forces F_{tx} and F_{rx} are equal, the following linearized equation applies to the rolling ball:

$$M * g * \alpha = \frac{2}{5} M \frac{d^2x}{dt^2}, \tag{5.8}$$

or in a state-space form:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{5}{7}g \end{bmatrix} u \tag{5.9}$$

Now, it can be seen that for the system identification, there is not much a difference between Eq. 5.4 and Eq. 5.9 when talking about linearized models. Another

approach is to derive the model from Lagrangian or Newtonian mechanics [4], controlling the position not directly by the beam angle α, but the torque applied to the beam, giving the state-space representation as:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\alpha} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-mgd}{(\frac{J}{R^2}+m)} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \alpha \\ \dot{\alpha} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u$$

Note that for the limited dimensions of the BoBShield, the torque can be substituteed with the angle multiplied by a constant in the process of identification.

5.2 Identification

Having determined the approximate form of the model we are looking for suggests that we will focus on gray box system identification. As it is known from previous studies, gray box identification is one of the three basic types of system identification beside the white box and black box approach. While the black box system identification requires basically no knowledge about the variables of the system and the relationships between them, on the contrary, white box modeling assumes that all variables and their mutual effects are known. Gray box modeling combines some advantages of both. In this case some specific form of equations describing the system is expected– as in white box approach, but knowing that not all constants appearingin these equations can be quantified. Therefore, in the gray box model, these constant are estimated based on some data from the system. Meaning, that before this a very important step should be done: the data acquisition. Although this may seem easy, one has to keep in mind that the quality of the system identification highly depends on its inputs. Therefore. it is essential to properly design the input signal. This signal must be designed in such a way that the open loop measured data will represent the characteristics of the system. For the specifics of the BoBShield system it was expected that the use of pseudo-random binary sequence signal (PRBS) can do the service but based on previous experiences from previous works [**?**] an amplitude-modulated pseudo-random signal (APRBS) would be more suitable for a non-linear system. Based on this knowledge, the APRBS signal had been chosen. The used APRBS signals was generated in MATLAB software environment using the `prbsGenerte` script, which is a part of the AutomationShield library. The first version of APRBS input signal was generated to change pseudo-randomly within the limits set as the minimal input angle at -30 degrees and the maximal input angle at 30 degrees. The input signal and the measured response is shown in Fig. 5.2. These results are chosen as a part of a longer experiment lasting 100 seconds, and the shown section is choosed as a part just right "long enough" with the minimalisation of the situations, where the ball is stuck in one or the other end of the tube for "too long".

As it is seen, even there was an effort to use a part of the whole experiment, where the ball shifts freely from end to end, this data can be hardly used for system identification. The problem is that in such setup the ball reaches the end of the tube

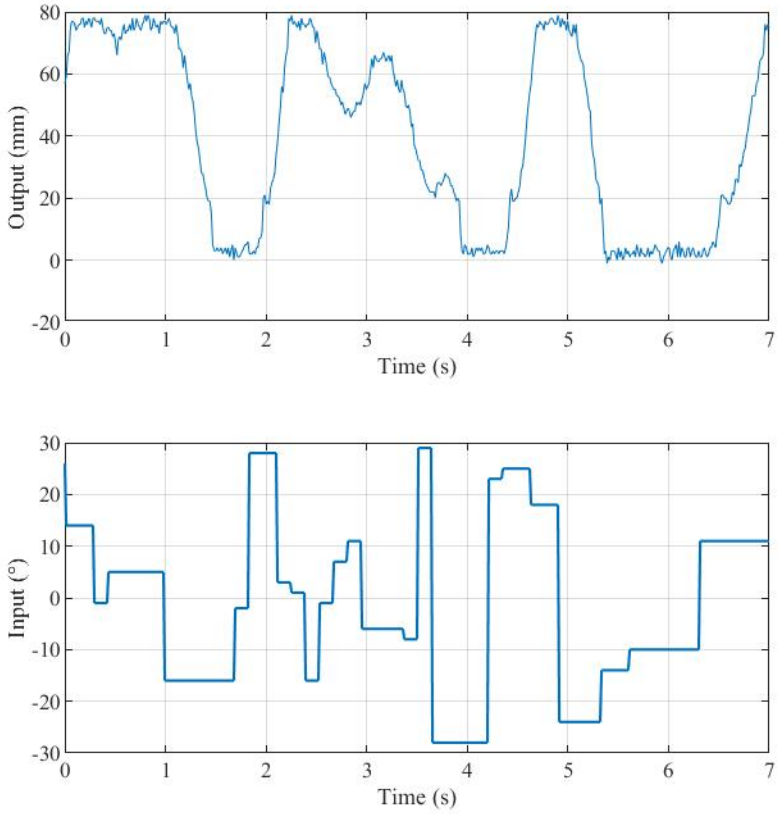


Figure 5.2: Open-loop response for system identification.

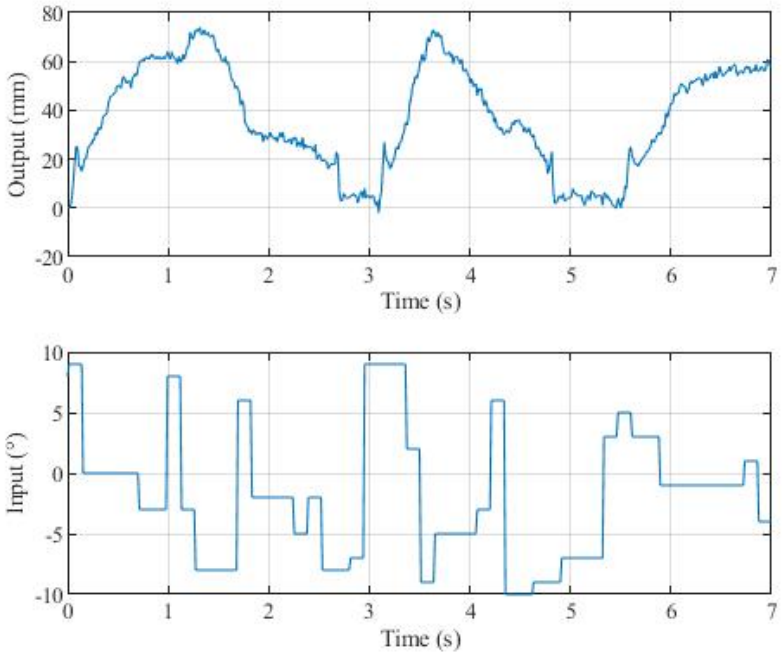


Figure 5.3: Open-loop response for system identification.

too often end gets stucked there, causing the data to seem obtained from a bounded, or even an assymptotically stable system. Which the ball on beam system is not, indeed. Here a dead end is reached, as there is attemption to model an unlimited system on a limited hardware. The solution is to change the input signal in a way, that the data does not get bounded by the hardware itself, or at least, the situation, where the ball hits the end of the tube does not last for too long. Therefore another input signal was generated, the same way as before, only the limits were changed to $\pm 10°$. The results from this version of experiment are shown in Fig. 5.3. Note that the problem mentioned above is at this moment still present, only its effects are reduced. A more accurate, but on the other hand way more complex solution could be to design the model in a form which somehow incorporates these hardware limitations.

With these data the identification itself can be finally started. For this, again the MATLAB software environment was used, more precisely the matlab identification toolbox. The whole process of identification is included in a MATLAB script in the appendix B. Firstly the data (both the input values and the measured distances) are loaded from the corresponding MATLAB file then converted to basic units (radians and meters). After that, from all the data a sequence with characteristic dynamics is selected. As a part of the data modification the function `detrend()` is used to them, removing the linear trend from the data. The function detrend subtracts the mean or a best-fit line (in the least-squares sense) from data. Removing a trend from the data enables the user to focus the analysis on the fluctuations in the data about the trend [18].In the next step, the known parameters of the system are specified, or more precisely, the expected and approximate values of these parameters. These

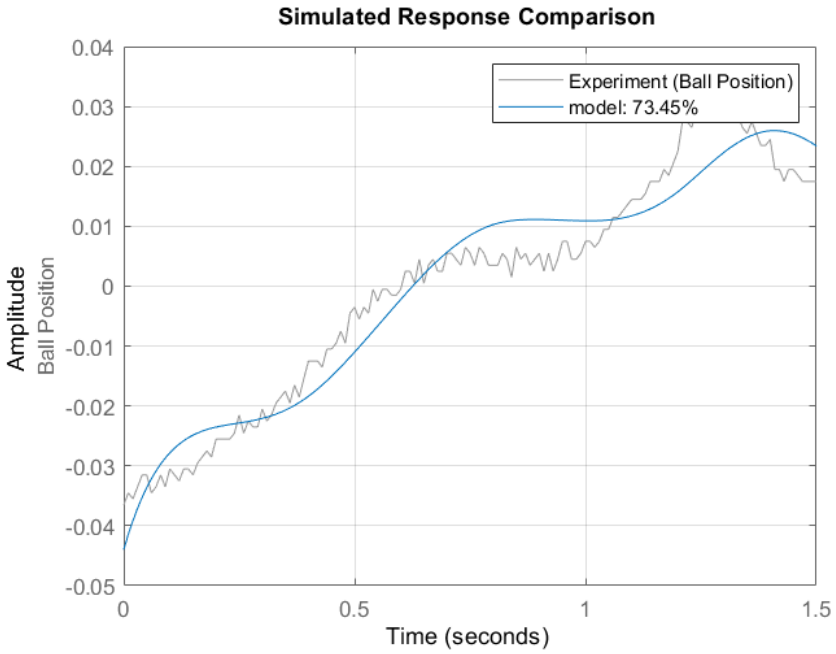


Figure 5.4: Transfer Function Identification: Simulated Response Comparison.

specified parameters will serve as the initial values for the estimation. At this moment, the user can chose a model, which is in the script realized via setting the values of variables `model` and `ODE`. Changing the variable `model` allows the user to switch between the types of gray-box model to identify:

- transfer function model,
- linear state-space model,
- non-linear model.

In the first case, when transfer function is chosen, the script uses the command `procest()` to find the best fitting model, with the results shown in Fig. 5.4.

In the second case, the model parameters are estimated with the help of the function `ssest()`, giving a linear state-space model based on the Eq. 5.4 or Eq. 5.9 with the results shown in Fig. 5.5.

The third option, using the `nlgreyest()` command, allows the user to specify the desired model in a form of any arbitrary (e. g. non-linear ordinary differential) equation. With this, various differential equations of the aforementioned mathematical models, which can possibly describe the ball on beam system were compared. The variable `ODE` is used in the script to select one of the possible models, which are implemented as MATLAB files. The results for the ODE given by Eq. 5.2 are in Fig. 5.6 and with the model from Eq. 5.10 in Fig. 5.7.

All results obtained from the identification are in line with the expectations based on the theory, as

- the least accurate model is the transfer function model,

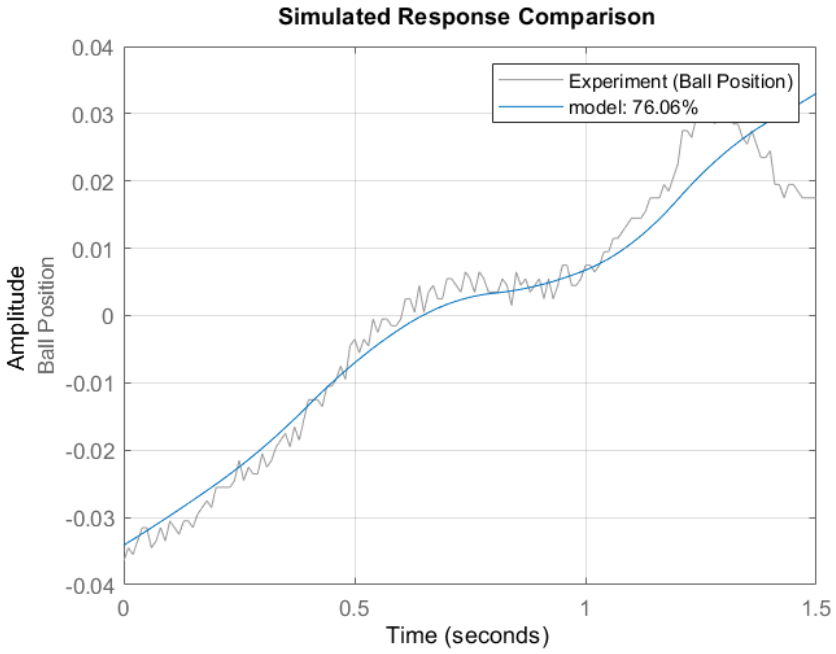


Figure 5.5: Linear Identification: Simulated Response Comparison.

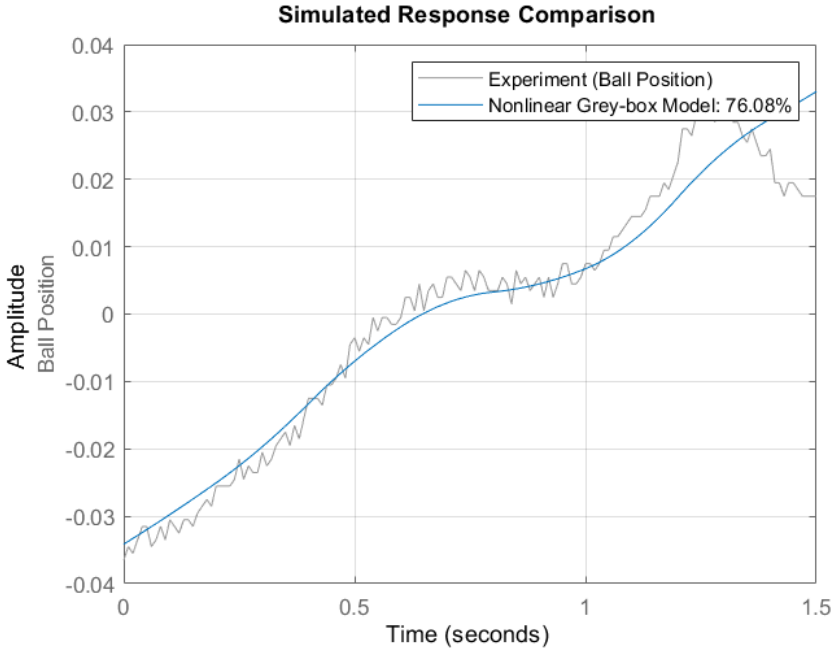


Figure 5.6: Non-linear Identification (2nd Order ODE): Simulated Response Comparison.

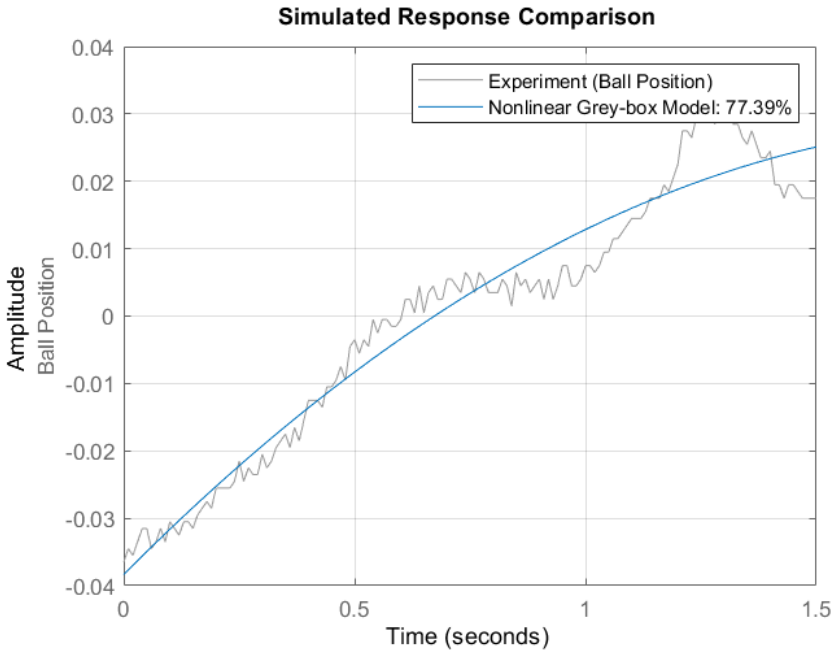


Figure 5.7: Non-linear Identification (4th Order ODE): Simulated Response Comparison.

- there is not much improvement using the non-model towards the linear one, giving the hardware congifuration, limitations and dimensions,
- even the parameter estimation based on Eq. 5.10 gives the best overall fit, based on the fact that in this case the torque was substituted with the angle multiplied by an estimated constant, this model does not fit the data properly in time.

Based on these observations, for the further work the continous time model obtained from the linear state-space identification is used, given as:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 4.93 \end{bmatrix} u \tag{5.10}$$

$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{5.11}$$

6 Examples

When discussing the currently available examples for the BoBShield device, this work will mainly focus on exampless in Arduino IDE, as these are the only finished and tested ones. Examples in other environments are currently being under developement, and will be later added to the AutomationShield project.

6.1 Self-test

The first example implements only a basic self-test routine, just to make the user sure everything works properly. After the inclusion of the two libraries needed, the VL1680X sensor is initialized, with a short test of functionality - if the micro controller unit cannot connect to the time of flight sensor, a failoure will be reported.

```
#include <BOBShield.h>                  // Includes all the
    necessary libraries nad hardware components
#include <SamplingServo.h>              // Include special
   Sampling running on timmer 2

Adafruit_VL6180X vl = Adafruit_VL6180X();
void setup() {

 Serial.begin(115200);                 // Initialize serial
 BOBShield.begin();
  // wait for serial port to open on native usb devices
  while (!Serial) {
    delay(1);
  }
  // finds out if sensor is active
  Serial.println("Adafruit VL6180x test!");
  if (! vl.begin()) {
    Serial.println("Failed to find sensor");
    while (1);
  }
  Serial.println("Sensor found!");
```

After the servo is turned to its minimal range and the distance is then measured, while displaying which part of the self-test is running. The measured distance is then tested, whether it is within the expected range. Note that in this example no

calibration is presented and the expected range for the maximal servo position is preset to $y \geq 80$ and $y \leq 100$. If these criteria are met, the message `Ok.` will appear in the serial line. In case of failure, the appearing message will be `Fail.`.

```
Serial.print("Turn servo to -30 degrees!  ");
Serial.print("\t");                 // print a tab
   character
Serial.print("\t");                 // print a tab
   character
Serial.print("\t");                 // print a tab
   character
BOBShield.actuatorWrite(-30);         // Tilt beam to
   the minimum so the ball falls towards the sensor
wait();                 // Wait for things to settle
int y = vl.readRange();
if (y >= 80 && y <= 100) {
  Serial.println(" Ok.");
}
else {
  Serial.println(" Fail.");
}
```

The same pattern is applied to the maximal servo range 30°. This part also serves to test whether the ball is present, as the measured position is expected to be ≤ 10, which can be achieved only where there is an object near to the sensor in the tube. After that, the last peripheral, the potentiometer is tested. At this part, the users action is requered, to turn the potentiometer, when needed. The user is instructed via the serial line. The obtained values are checked, if they are in the expected ranges.

```
// potentiometer test
Serial.print("Turn pot to 0%!  ");
Serial.print("\t");                 // print a tab
   character
Serial.print("\t");                 // print a tab
   character
Serial.print("\t");                 // print a tab
   character
wait();
float z = BOBShield.referenceRead();
if (z >= 0.0 && z <= 1.0) {
  Serial.println(" Ok.");
}
else {
  Serial.println(" Fail.");
}
Serial.print("Turn pot to 100%!   ");
Serial.print("\t");                 // print a tab
```

```
    character
  Serial.print("\t");                  // print a tab
    character
  Serial.print("\t");                  // print a tab
    character
  wait();
float  x = BOBShield.referenceRead();
  if (x >= 99.0 && x <= 100.0) {
    Serial.println(" Ok.");
  }
  else {
    Serial.println(" Fail.");
  }
*/
}
```

The following examples show the implementation of various control algorithms on the BoBShield hardware, using the Arduino API.

- PID
- LQ
- MPC

6.2 PID

One of the first control algorithms one may encounter while studying control theory is the proportional-integral-derivative controller (PID). A PID controller calculates the control action from the error value $e_{(t)}$ continously with the standard form of the equation given as:

$$u_{(t)} = K_p(e_{(t)} + \frac{1}{T_i}\int_0^t e_{(\tau)}\,d\tau + T_d\frac{de_{(t)}}{dt}), \tag{6.1}$$

where: $u_{(t)}$ is the calculated system input, $e_{(t)}$ is the error value, the difference between the reference and the system output at time t,

and as the controllers name suggests, K_p is the proportional gain, T_ii s the integral time, T_d is the derivative time.

Note that the formula above stands for a controller in continous time, for discrete time this is modified replacing the integral with a sum and the differencial with a difference as:

$$u_{(Tk)} = K(e_{(k)} + \frac{T}{T_i}\sum_{i=0}^{k-1} e_{(i)} + \frac{T_d}{T}[e_{(k)} - e_{(k-1)}]), \tag{6.2}$$

where: T is the sampling time, k is the index of the sample, $u_{(kT)}$ is the calculated system in the sample k. $e_{(k)}$ is the error value in the sample k. For the sampling time $T = 1$, the Eq. 6.2 will be simplified as:

$$u_{(k)} = K(e_{(k)} + \frac{1}{T_i}\sum_{i=0}^{k-1} e_{(i)} + T_d[e_{(k)} - e_{(k-1)}]), \tag{6.3}$$

This is often called as a PSD (proportional-summation-difference) controller, but it is also common to refer to this type of controller as PID too, so in this thesis only the term PID controller will be used, in the sense following the context.

In this example from the file `BOBShieldPID` available on GitHub [**?**] the application of the PID controller is provided. First the sampling time and the sampling index are declared. The sampling time had been choosen as 10 milliseconds. The reference vector `R` is defined by the user at the beginning of the script. Also a boolean variable `enable` is created and at the beginning set to the value `false`. This variable is later used to determine weather the `step()` function should be called in the `loop()`. The used PID controller is manually tuned to $K_p = 0.3(mm/degree), T_i = 600$ and $T_d = 0.3$

```
#include <BOBShield.h>
#include <SamplingServo.h>

unsigned long Ts = 10;              // Sampling in
   milliseconds
unsigned long k = 0;                // Section index
bool enable=false;                  // Flag for sampling

float r = 0.0;                      // Reference
float R[]={30.0,50.0,8.0,65.0,15.0,40.0};;    //
   Reference trajectory

int T = 1000;             // Section length
int i = 0;                // Section counter
float y = 0.0;              // Output
float u = 0.0;              // Input

#define KP 0.3                      // PID Kp
#define TI 600.0                    // PID Ti
#define TD 0.22                     // PID Td
```

In the `setup()` function besides the shield initialization the PID constants are assigned and the Serial port is "turned on"

```
void setup() {
  Serial.begin(115200);                 // Initialize
     serial

  // Initialize and calibrate board
```

```
  BOBShield.begin();                // Define hardware
     pins
  BOBShield.initialize();
  BOBShield.calibration();

  // Initialize sampling function
  Sampling.period(Ts * 1000);   // Sampling init.
  Sampling.interrupt(stepEnable); // Interrupt fcn.

 // Set the PID constants
 PIDAbs.setKp(KP);
 PIDAbs.setTi(TI);
 PIDAbs.setTd(TD);
}
```

In the main loop, based on the interrupt service routine (ISR) the `step()` function is called, when needed.

```
void loop() {
  if (enable) {                 // If ISR enables
    step();                   // Algorithm step
    enable=false;                 // Then disable
  }
}
```

The function `step()` contains all operations, that should be performed in one sample:

- The actual reference value is obtained by assigning the one element of the reference vector `R` to variable `r` when needed (based on the sequence length);
- The measurement is done and assign the actual ball position and assign it into the variable `y`. As the PID controller is a very simple algoritm, which does not require the knowledge of the model itself and uses simple mathematical operation, there is no need to make the measurements any more precise, e.g. using a Kalman filter, the measurements are used directly in the further calculations;
- Calculation of the system input `u` is provided by calling the function `PIDabs.compute()`. This function is a part of the AutomationShield library and proviedes the calculation using the 6.2 from the error value `r-y`. This function also takes care of limiting the calculated input within the defined range -30 +30 degrees. This means that the calculated value of `u` is saturated before passing it out of the function;
- Aplying the calculated input value on the servo;
- Sending the reference, output and the input to the serial port;

Figure 6.1: Screenshot of a PID experiment in the Arduino IDE Serial Plotter.

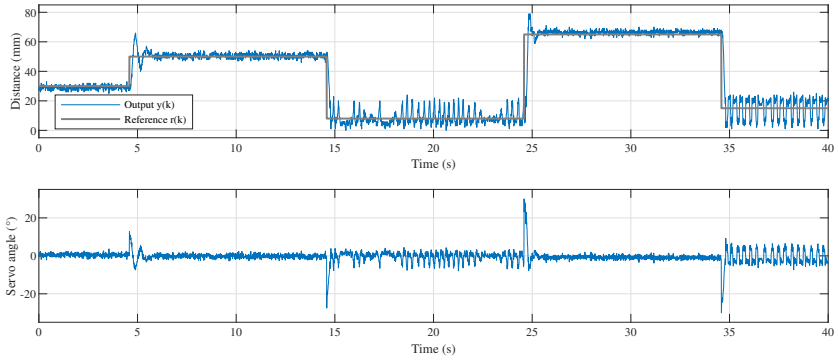


Figure 6.2: PID control of the ball position.

```
void step(){                        // A single algorithm step
if (k \% (T*i) == 0){                // End of reference section
  r = R[i];                         // Set reference
  i++;                              // Increment reference counter
}
// Measure, compute, actuate:
y = BOBShield.sensorRead();                  // Read sensor
u = PIDAbs.compute(r-y,-30,30,-10,10);       // PID
BOBShield.actuatorWrite(u);                  // Actuate
```

The outputs of the program then can be viewed in the Arduino IDE Serial Plotter as shown in Fig. 6.1., where the process variabler `r`, `y` and `u` labelled with blue, red and green respectively. The outputs of the whole example are shown plotted in MATLAB in Fig. 6.2.

6.3 LQ

Although the biggest advantage of PID control lies in its simplicity, easy implementation and possibility to set parameters using various automatic tuning techniques, it naturally has considerable limitations. Therefore, if one wants to obtain a better control, it is better to look for some other method. Fortunately, a wide range of other control algorithms is availiable. Control algorithms based on optimalization of the control give qualitatively better results than traditional algorithms based only on manipulation with the transfer. this means that these algorithms follow the predefined trajectory more precisely, or they react to extern disturbances in a shorter time. These algorithms can cope with multidimensional systems and unautonomous systems with link between input and output. The well-known linear-quadratic (LQ) regulator can be included into the class of control algorithms. The LQ regulator is based on the minimization of the quadratic cost function on an infinite horizont meaning that the optimal system input $u(k)$, that minimizes the quadratic cost $J(k)$ given as 6.4 is searched for.

$$J_{(k)} = \sum_{k=1}^{\inf} (\mathbf{x}_{(k)}^T \mathbf{Q} \mathbf{x}_{(k)} + \mathbf{u}_{(k)}^T \mathbf{R} \mathbf{u}_{(k)}), \tag{6.4}$$

where: $J_{(k)}$ is the quadratic cost, $\mathbf{Q}$ is the state penalty matrix, $\mathbf{R}$ is the input penalty matrix,

$\mathbf{x_{(k)}}$ is the state vector, $\mathbf{u_{(k)}}$ is the input vector [8].

As it can be seen, the LQ regulator, unlike the PID controller, incorporates a bit more knowledge of the system, namely, the state-space model of the system is required, in form of:

$$\mathbf{x}_{(k+1)} = \mathbf{A}\mathbf{x}_{(k)} + \mathbf{B}\mathbf{u_{(k)}}, \mathbf{y}_{(k)} = \mathbf{C}\mathbf{x}_{(k)} + \mathbf{D}\mathbf{u_{(k)}}, \tag{6.5}$$

where $\mathbf{y}$ is the system output vector at discrete time k,
$\mathbf{A}$ is the system matrix,
$\mathbf{B}$ is the input matrix,
$\mathbf{C}$ is the output matrix,
$\mathbf{D}$ is the feedthrough matrix.

The control law of the LQ regulator is given very simply, as

$$\mathbf{u}_{(k)} = -\mathbf{K}\mathbf{e}_{(k)} \tag{6.6}$$

where $\mathbf{e}_{(k)}$ is the difference between the state vector and the reference state vector and K is the gain matrix. Even this control law does not seem to be any complicated or sophisticated, its power lies in the process how the gain matrix is calculated, that is by solving the algebraic Riccati equation given in the form:

$$\mathbf{P}_{(k)} = (\mathbf{A} + \mathbf{B}\mathbf{K}_{(k)})^T \mathbf{P}_{(k)} (\mathbf{A} + \mathbf{B}\mathbf{K}_{(k)}) + \mathbf{K}_{(k)}^T \mathbf{R} \mathbf{K}_{(k)} + \mathbf{Q}, \tag{6.7}$$

$$\mathbf{K}(k) = (\mathbf{R} + \mathbf{B}^T \mathbf{P}_{(k)} \mathbf{B})^{-1} \mathbf{B}^T \mathbf{P}_{(k)} \mathbf{A}, \tag{6.8}$$

where $\mathbf{P}_{(k)}$ is terminal state penalty matrix at a discrete time k. Note that both these equations are interdependent and must be solved at the same time. Again, there are existing algorithms to solve the algebraic Riccati equation and fortunately they are implemented in MATLAB, which is used in this thesis to calculate the gain matrix K by using the `dlqr()` function.

6.3.1 Trajectory Following with LQ - Integrator State

At this point, it had to be said, that in general, LQ regulator is considered to be better than a simple PID controller, however, has no option of monitoring the past errors by default. This can lead to some problems, e.g. if for any reason the LQ regulator cannot calculate high enough or low enough input a stady-state error can be achieved. Especially when the system is not regulated into zero point (or any constant value), but to follow a path, this problem can arise. It is therefore necessary to somehow eliminate this shortage. This is done by the introduction of the integrator state.

As the word state suggests, the aforementioned monitoring of past errors is added to the model as another state. Since the aim of the integrator state is simply defined as a vector of differences between the reference and the output summarized in the time:

$$\mathbf{x}^I_{(k+1)} = \mathbf{x}^I_{(k)} + (\mathbf{r}_{(k)} - \mathbf{y}_{(k)}), \tag{6.9}$$

or

$$\mathbf{x}^I_{(k+1)} = \mathbf{x}^I_{(k)} + (\mathbf{r}_{(k)} - \mathbf{C}\mathbf{x}_{(k)}). \tag{6.10}$$

Inclusion of the integrator state is then done manually, outside the controller in every iteration. The augmented state vector then is defined as:

$$\begin{bmatrix}\mathbf{x}_{(k+1)}\\ \mathbf{x}^I_{(k+1)}\end{bmatrix} = \begin{bmatrix}\mathbf{A} & \mathbf{0}\\ \mathbf{C} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{x}_{(k)}\\ \mathbf{x}^I_{(k)}\end{bmatrix} + \begin{bmatrix}\mathbf{B}\\ \mathbf{0}\end{bmatrix}\mathbf{u}_{(k)} + \begin{bmatrix}\mathbf{0}\\ \mathbf{I}\end{bmatrix}\mathbf{r}_{(k)}. \tag{6.11}$$

The model extended by the integrator state is then given by:

$$\tilde{\mathbf{x}}_{(k+1)} = \tilde{\mathbf{A}}\tilde{\mathbf{x}}_{(k)} + \tilde{\mathbf{B}}\mathbf{u_{(k)}} + \mathbf{r_{(k)}}, \tag{6.12}$$

where

$$\tilde{\mathbf{x}}_{(k)} = \begin{bmatrix}\mathbf{x}_{(k)}\\ \mathbf{x}^I_{(k)}\end{bmatrix}, \tilde{\mathbf{A}} = \begin{bmatrix}\mathbf{A} & \mathbf{0}\\ \mathbf{C} & \mathbf{I}\end{bmatrix}, \tilde{\mathbf{B}} = \begin{bmatrix}\mathbf{B}\\ \mathbf{0}\end{bmatrix}.$$

The equation given by 6.11 is the new, extended model, which will be used to obtain the extended gain $\tilde{K}$. Naturally, the impact of the integrator state can be set by changing the respective element of the extended penalty matrix $\tilde{\mathbf{Q}}$.

As one may notice, in this case, the tuning of the regulator is not as obvious as in the case of PID, it only happens by choosing the right matrices in the cost function. Usually in publications one can find that these matrices are chosen optionally, only the matrix $\mathbf{Q}$ has to be a positive semidefinite matrix and the $\mathbf{R}$ must be a positive

definite matrix. One may also find that matrix $\mathbf{Q}$ is often given as $\mathbf{Q} = \mathbf{C}^T * \mathbf{C}$. However, for the ball and beam system these principles are less than enough. After having some difficulties tuning these matrices, another optimization algorithm was chosen to do so, namely the genetic algorithm.

6.3.2 LQ Tuning - Genetic Algorithm

The genethic algorithm (GA) is not an algorithm particularily designed to find the optimal values of matrices $\mathbf{Q}$ and $\mathbf{P}$, but as an algorithm used for solving optimization problems in general, it can be used for that too. The idea of this algorithm lies in mimicking the biological evolution, meaning that the algorithm repeatedly modifies a population of individual solutions. The algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm randomly selects individuals from the current population and uses them as parents to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution.(MATLAB) The biggest differences between the genetic algoritm and the classical, derivative-based optimization algorithm lies in [19]:

- point generation in each iteration
- and the selection of the next point.

While the classical algortihm generates a single point at each iteration, the genetic algorithm generates a whole set, called population, of them. Meaning, that the genetic algorithm the best point in the population approaches an optimal solution in comparison with the sequence of points approaching the optimal solution of classical algorithm. Also, the genetic algorithm as a probabilistic method uses random number generators to select the next population, in contrast to next point selection by a deterministic computation.

In this term, the population is the set of variables to be chosen (for us, the values on the diagonal of matrices $\mathbf{Q}$ and $\mathbf{R}$, while the optimal solution is obtain from the objective function. The objective function `gaTune.m` is created as the norm of the error value vector trough an LQ simulation `LQtune.m`.

After the modification with the integrator state, the optimal matrices $\mathbf{P}$ and $\tilde{\mathbf{Q}}$ are obtained using the MATLAB script `gaTune.m` and then the extended gain $\tilde{K}$ is calculated using the function `dlqr()`.

Before stepping to the Arduino IDE a simple LQ regulation was made using the gain vector $\tilde{K}$ obtained as described above with the code in MATLAB as:

```
for k=1:N

    U(k)=-K*[X(:,k); XI]; % system input computed with
       the state vector extended by the integration state
    %system input constraints
          if U(k)>deg2rad(5);
        U(k)=deg2rad(5);
      elseif U(k)<-deg2rad(5);
```

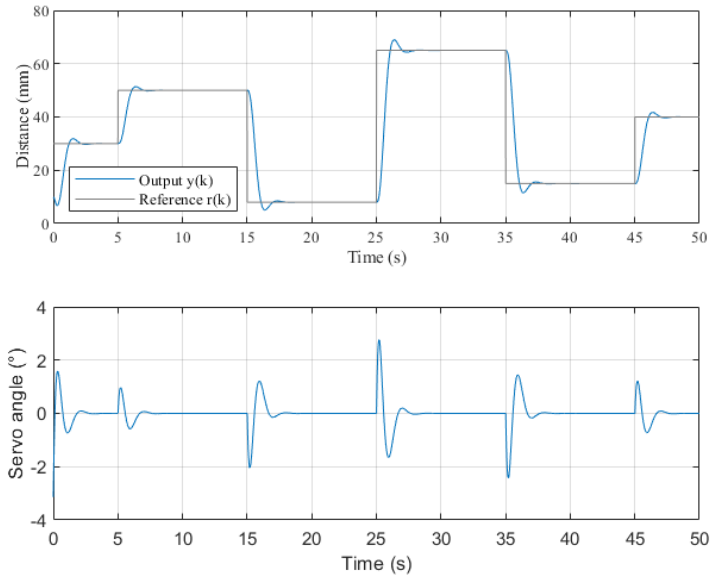


Figure 6.3: LQ simulation in MATLAB.

```
        U(k)=-deg2rad(5);
    end;

    X(:,k+1)=A*X(:,k)+B*U(k); % Real system
    Y(:,k)=C*X(:,k); % Real system
    XI = XI + (R(k)-Y(k)); % Integration state
end
```

The results of this simple simulation are shown in the figure 6.3.2. Regarding the implementation of this example in Arduino IDE, the steps will be very similar to the PID example. Firstly, the constant and variables are declared, on the top of that the states-space model of the system, the process and measurement noise covariance have to be added too. This is needed only because, in this example not the raw measurements of the ball position are used, but the Kalman estimation of the state, as the LQR needs to have every state known. The Kalman estimation of the state is obtained from the function `getKalmanEstimate();`, which is a part of the AutomationShield library.

```
//Defining State-Space matrices of the system
BLA::Matrix<2, 2> A = {1.0, 0.01, 0.0,  1.0}; // State
   matrix A
BLA::Matrix<2, 1> B = {0.0002, 0.0422};
                                    // Input matrix B
BLA::Matrix<1, 2> C = {1.0, 0.0};
                                                    //
   Output matrix C

// Kalman process and measurement error covariances
```

```
BLA::Matrix<2, 2> Q_Kalman = {100, 0.0,0.0,1.0};     //
   Process noise covariance matrix
BLA::Matrix<1, 1> R_Kalman = {100};
                                      // Measurement noise
   covariance matrix

// LQ gain with integrator
BLA::Matrix<1, 3> K = {5.7589, 1.6407, -0.0965};
   // Pre-calculated LQ gain K
BLA::Matrix<2, 1> X = {0.0, 0.0};
                                     // Estimated state
   vector
BLA::Matrix<3, 1> XI = {0.0, 0.0, 0.0};
                                     // Estimated state
   vector including integrator state
BLA::Matrix<3, 1> Xr = {0.0, 0.0, 0.0};
                                    // Reference state
   vector
```

The `step()` is defined similarily, only the aformentioned Kalman estimation of the states is added as:

```
void step(){                  // A single algorithm step
if (k \% (T*i) == 0){          // End of reference section
  r = R[i];                   // Set reference
  i++;                        // Increment reference counter
}
// Measure, compute, actuate:
y = BOBShield.sensorRead();            // Read sensor
  u =( -(K * (XI - Xr))(0));
                                                        //
     Calculate LQ system input
  u=rad2deg(u);
  if (u>30.0) {u=30.0;}
  if (u<-30.0) {u=-30.0;}
  BOBShield.actuatorWrite(u);
                                                    // Actuate
  getKalmanEstimate(XI, deg2rad(u), y, A, B, C, Q_Kalman,
      R_Kalman);    // Estimate internal states X
  XI(2) = XI(2) + (Xr(0) - XI(0));
                                                    // Add
     error to the summation state
}
```

6.4 MPC

Both control algorithms mentioned above are good enough to control the system and as stated before, the LQ is more sufficient than the PID but both have disadvantages such as:

- they do not contain explicitly, within the criterion function, set objectives of the control,
- they do not consider the limitations arising from the process, or technology,
- they do not behave robustly against accidental changes and faults,
- they focus on the effects of the controlling only at a local level,
- they focus only on achieving short-term goals. [8]

A better controlling can be achieved by optimization of the control troghout the entire time of controlling, which is the base idea of the model predictive control (MPC). With this algorithm in each step the best current step is calculated, which will lead to the optimal outcome in the future. In addition, the model predictive control is not just better in terms of following the reference value, but it also brings the opportunity of implementation the constraints mathematically and calculate the optimal input with the respect of them. Using MPC constraints can be applied not just on the system inputs, but on every state considered in the model. As the whole mathematical concept is not in the scope of this thesis, let the equation describing the problem solved in model predictive control be introduced:

$$\overrightarrow{\mathbf{u}}^*(k) = \underset{\overrightarrow{\mathbf{u}}(k)}{\operatorname{argmin}} \left(\frac{1}{2} \overrightarrow{\mathbf{u}}^T(k) \mathbf{H} \overrightarrow{\mathbf{u}}(k) + \mathbf{x}^T(k) \mathbf{G}^T \overrightarrow{\mathbf{u}}(k) \right), \tag{6.13}$$

where $\overrightarrow{\mathbf{u}}^*(k)$ is the optimal sequence of system inputs at discrete time k,
$\overrightarrow{\mathbf{u}}(k)$ is the sequence of system inputs at discrete time k,
$\mathbf{x}(k)$ is the state vector at discrete time k,
$\mathbf{H}$ is the Hessian matrix and
$\mathbf{G}$ is the matrix of MPC cost function, while the expression $\mathbf{x}^T(k)\mathbf{G}^T$ is the gradient of the cost function.

The Eq. 6.13 is nothing else than the mathematical expression of the main goal od MPC as described before: to find a sequence of system inputs minimizing the cost function in Eq. 6.14. Note that the cost function on the right of Eq. 6.13 does not containt the expression $\mathbf{x}^T(k)\mathbf{F}\mathbf{x}(k)$; that is because this element of the cost function has no effect on the optimised variable, so it can be omitted.

$$J(k) = \mathbf{u}^T(k)\mathbf{H}\mathbf{u}(k) + 2\mathbf{x}^T(k)\mathbf{G}\mathbf{u}(k) + \mathbf{x}^T(k)\mathbf{F}\mathbf{x}(k), \tag{6.14}$$

Now the constraints are given in a form of inequations

$$\mathbf{A}_c \overrightarrow{\mathbf{u}}(k) \leq \mathbf{b}_0 + \mathbf{B}_0 \mathbf{x}(k). \tag{6.15}$$

Through this inequation, the system input constraints (using only the utilising matrices $\mathbf{A}_c$ and $\mathbf{b}_0$) or the system state constraints (with using the utilising matrix $\mathbf{B}_0$ too) can be implemented.

Generally, matrices $\mathbf{H}$ and $\mathbf{x}^T(k)\mathbf{G}^T$ are the mandatory inputs for a solver, the task of which is to find the oprimal $\overrightarrow{\mathbf{u}}^*(k)$ and the utilising matrices are used only when constraints are present.

Though it may seem that the MPC is the very best solution for controlling, it has also its disadvantages, namely the computational complexity, or complexity. As the magic of MPC lies in "seeing the future", in sense of some kind of simulations using the model of the system, one may naturally expect a lot of matrices during the calculations. As a result, this control algorithm performs a several times more calculations in each step, than LQ regulator, not to mention the PID controller. Having a suitable MCU and sufficient memory, this is not a problem, but when implementing such an algorithm on a limited hardware as Arduino, some issues may arise. As the implementation of the *μAO-MPC* [22] package, for solving the Eq. 6.13 in Arduino IDE environment has not been completed by now, a simple simulation experiment was conducted in MATLAB to show (Fig. [**?**]) how the prediction horizont affects the quality of control. As expected, the ball on beam system can be conveniently regulated with a horizont just 4 steps ahead, resulting in matrices, that may be handlable by the hardware. For this simulation a simple script in MATLAB was created, using the model and reference vector used in the LQ example. Similarly to LQ, the integrator state was incorporated to the computation of the system input. The boundaries given by the hardware configuration were set, and lastly, the process of control was simulated for various length of the horizont. For solving the Eq. 6.13 the MATLAB `quadprog()` function in the way described in the curse of Theory of Automatic Control, with examples available on GitHub [17].

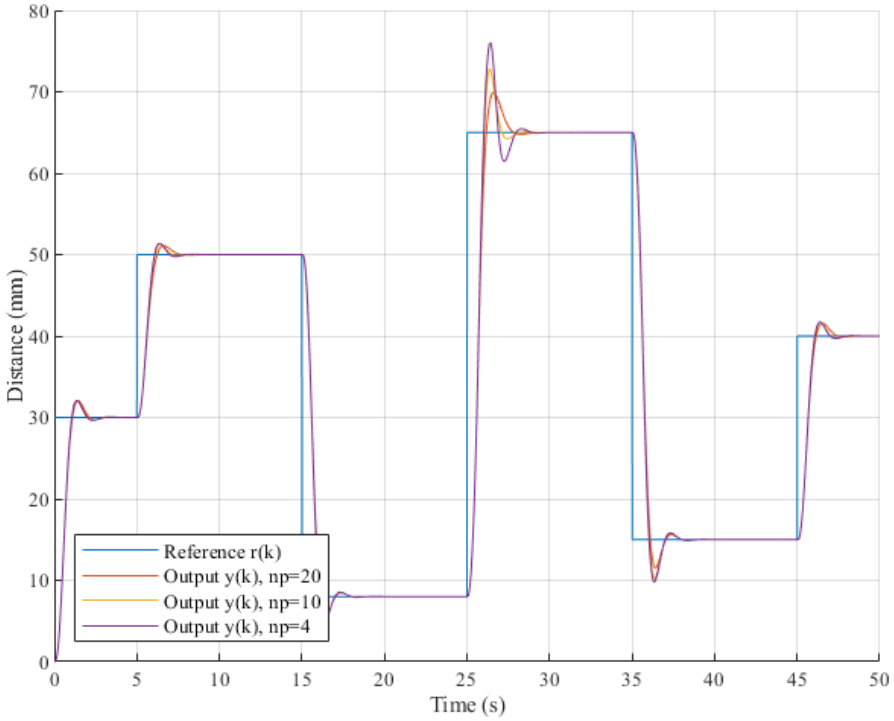


Figure 6.4: LQ simulation in MATLAB.

7 Conclusion

In this thesis, a brief introduction to the topic was provided, with a short review of works dealiing with the ball and beam experiment, either theoretically, or focusing on hardware design and implementation of control algorithms. As this particular thesis deals with the BoBShiels device, as a part of the AutomationShield initiative, the main goals and achievements of this initiative were also introduced. Focusing on the BoBShield, the basic concept was introduced and the revision of the first version of the hardware, created within the boundaries of Control Theory curse, was given. After that, the changes implemented for the second version of the hardware were described. As the BoBShield device will be used to teach technical disciplines in the general field of automation, application programming interfaces in environments Arduino IDE and Simulink were introduced as well. The underlying dynamic phenomenon of the experiment was described, with a suitable test signal designed for identification. The unknown parameters were identified utilizing MATLAB. Finally, didactic tasks for the device were created in the Arduino IDE application programming interface, implementing a short self-test, PID control, and LQ regulator, with the changes and improvements recorded and managed using GitHub.

Even in these difficult "Corona Times", the work on and with such a compact and portable device seemed to be a great way to being able to do practical research instead of something more theoretical, the current situation left a mark on the progress of this thesis. Limited access to laboratory device, issues with the logistics and lack of personal meetings resulted in a lot of work to be done, before the BoBShield can be considered to be ready for teaching. Nevertheless, both the thesis supervisor and the consultat did their best to help the author.

In further work it would be great to create an API in other environments, such as MATLAB and Python. Also the didactic tasks should be implemented in other environments and maybe expand the range of available examples with the use of other control algorithms, e.g. adaptive PID or pole placement. On top of that, it is not recommended to consider the current version of the hardware as a final, as with another configuration new opportunities can arise.

Bibliography

[1] . Ball & beam: System modeling. User's manual. Online., 2021. [cit. 2021-05-18] https://forums.ni.com/t5/myRIO-Student-Projects/myBall-Beam-Classic-Control-Experiment/ta-p/3498079?profile.language=en.

[2] A. Tota. Ball and beam v2, 2021. [cit. 2021-05-18] https://www.daslhub.org/unlv/wiki/doku.php?id=lego_ball_and_beam_v2.

[3] K. Benchikha. Ball and Beam: PID control on Arduino with LabView to stabilize a ball on a beam. Online, 2019. [cit. 2021-05-18]; Available from https://create.arduino.cc/projecthub/karem_benchikha/ball-and-beam-601d7a. Arduino Project Hub.

[4] C. G. Bolívar-Vincenty and G. Beauchamp-Báez. Modelling the ball-and-beam system from newtonian mechanics and from lagrange methods. In *Twelfth LACCEI Latin American and Caribbean Conference for Engineering and Technology (LACCEI'2014) "Excellence in Engineering To Enhance a Country's Productivity" July 22 - 24, 2014 Guayaquil, Ecuador.*, 2014.

[5] T. Emami. Mixed sensitivity design of pid controller-applied to a ball and beam system. In *ASME 2017 International Mechanical Engineering Congress and Exposition*, volume 4. ASME, 2017.

[6] N. Eqra, R. Vatankhah, and M. Eghtesad. A novel adaptive multi-critic based separated-states neuro-fuzzy controller: Architecture and application to chaos control. *ISA Transactions*, 2020.

[7] I. D. H.-G. Erick P. Herrera-Granda. Controller comparison and mathematical modelling of ball and beam system. [cit. 2019-04-07] http://revistasojs.utn.edu.ec/index.php/ideas/article/view/349, 2020.

[8] M. G. G. Takács. *Základy Prediktívneho Riadenia.* Slovenská technická univerzita v Bratislave, Bratislava, 2018.

[9] G. Takács, E. Mikuláš, A. Vargová, et. al. Bobshield: An open-source miniature "ball and beam" device for control engineering education. In *2021 IEEE Global Engineering Education Conference (EDUCON)*, 2021.

[10] Z. Gao, S. Wijesinghe, T. Pathinathanpillai, E. Dyer, and I. Singh. Design and implementation a ball balancing system for control theory course. 2015.

[11] N. Hara, M. Takahashi, and K. Konishi. Experimental evaluation of model predictive control of ball and beam systems. In *Proceedings of the 2011 American Control Conference*, pages 1130–1132, 2011.

[12] Internet Movie Database. Robert Shield. [cit. 2021-05-17] `https://www.imdb.com/name/nm0793158/`, 2021.

[13] Quanser. Ball and beam. online, 2021. [cit. 2021-05-25]. `https://www.quanser.com/products/ballandbeam/`.

[14] F. A. Salem. Mechatronics design of ball and beam system: Education and research. *Control Theory and Informatics*, 3(4), 2013.

[15] S. Sameera, R.; Arun. Position tracking of ball and beam system using fractional order controller. *International Journal of Advanced Research Trends in Engineering and Technology*, 4(6):174–180, 2017.

[16] G. Takács, P. Chmurčiak, R. Koplinger, T. Konkoly, G. Penzimger, M. Gulan, J. Kulhánek, L. Vadovič, M. Biro, E. Vargová, M. Vríčan, and J. Mihalík. AutomationShield. `https://github.com/gergelytakacs/AutomationShield`, 2021.

[17] G. Takács and M. Gulan. Základy Prediktívneho Riadenia. `https://github.com/gergelytakacs/Zaklady-prediktivneho-riadenia`, 2018.

[18] The Mathworks. Ball and beam: System modeling. User's manual. Online., 2021. [cit. 2021-04-07] `https://www.mathworks.com/help/matlab/data_analysis/detrending-data.html`.

[19] The Mathworks. Ball and beam: System modeling. User's manual. Online., 2021. [cit. 2021-04-07] `https://www.mathworks.com/discovery/genetic-algorithm.html`.

[20] University of Michigan. Control system toolbox. User's manual. Online., 2021. [cit. 2021-04-07] `https://ctms.engin.umich.edu/CTMS/index.php?example=BallandBeamsection=SystemModeling`.

[21] M. Virseda. *Modeling and Control of the Ball and Beam Process*. PhD thesis, Lund Institute of Technology, Lund, Kingdom of Sweden, March 2004.

[22] P. Zometa, M. Kögel, and R. Findeisen. muAO-MPC: a free code generation tool for embedded real-time linear model predictive control. In *Proc. American Control Conference (ACC), 2013*, pages 5340–5345, Washington D.C., USA, 2013.

Resume

Záverečná práca je zameraná na miniaturizovaný experiment „guľôčka na tyči" („ball on beam"). V úvode je v krátkosti zhrnutá dôležitosť hardvérových prvkov, ktoré pomáhajú pochopiť a študovať niektoré fyzikálne javy. Hlavne v štúdiu niektorých zákonitostí existuje niekoľko známych experimentov, ktoré sa často pozorujú a analyzujú, medzi ktoré patrí aj experiment skúmaný v tomto texte. V prvej kapitole sa uvádza prehľad dostupnej literatúry na túto tému, pričom vyplývajúc z popularity experimentu, nie je núdza o články, vedecké a záverečné práce, ktoré sa zaoberajú guľôčkou na tyči. Tieto práce spracovávajú tému či už na teoretickej úrovni, odvodením matematického modelu, prípadne so simuláciou pohybu, alebo priam simulovaním aplikácie riadiaceho algoritmu na tento systém. Iné na druhej strane opisujú aj dostupný hardvér s aplikáciami riadiacich algoritmov a dokonca existujú zdroje, ktoré poskytujú návod na vytvorenie vlastného zariadenia.

V ďalšej kapitole sa pristupuje k opisu konkrétneho zariadenia. Zariadenie nesie názov BoBShield, z anglického slova „shield", ktoré v tomto prípade označuje dosku plošných spojov kompatibilnú s prototypizačnými doskami Arduino a zo skratky anglického spojenia „ball on beam". Zariadenie je súčasťou iniciatívy AutomationShield, ktorej cieľom je vytvorenie zariadení využiteľných vo výučbe teórií riadenia, spolu s rozhraním pre programovanie aplikácií a didaktickými príkladmi v prostrediach ako je Arduino IDE, MATLAB, Simulink a Python. Napriek širokej škále komerčne dostupných zariadení určených na výučbu a výskum, jedinečnosť hardvérov obsiahnutých v projekte AutomationShield vyplýva z ich dostupnosti, či už čo sa týka hardvérového dizajnu alebo softvéru a z toho, že na ich vytvorenie v domácich podmienkach nie sú potrebné žiadne špeciálne súčiastky. Navyše, svojimi rozmermi a kompaktnosťou poskytujú možnosť skúmať nimi reprezentované javy aj v domácom prostredí, bez nutnosti prístupu k dobre vybavenému laboratóriu.

Experiment reprezentovaný zariadením BoBShield sa kladá z guličky (nemusí to byť nevyhnutne guľa, ako je na to poukázané v Kap. 1), ktorá sa kotúľa pozdĺž podporného povrchu, v tomto prípade priehľadnej plastovej trubky, v ktorom sa nachádza. Táto trubka je namontovaná na servo motor, pričom zo všetkých možností spojenia trubky s motorom, na skúmanom zariadení trubka pripevnená priamo na os motora, v strede svojej dĺžky. Vstupom do systému je potom uhol natočenia osi motora. Úlohou riadenia tohto experimentu je stabilizovať guličku v danej polohe. Poloha je odmeriavaná pomocou TOF senzora VL6180X, umiestneného na jednom konci trubičky. BoBShield ešte obsahuje potenciometer, pre manuálne nastavenie referenčnej hodnoty.

Prvá verzia hardvéru bola vytvorená študentmi predmetu Mikropočítače a mikro-

procesorová technika na Strojníckej fakulte Slovenskej Technickej Univerzite v Bratislave, hoci tento prvý návrh bol plne funkčný, niektoré jeho nedostatky boli odstránené pri vývoji druhej verzie. Tieto nedostatky sa hlavne týkali hlavne rozloženia jednotlivých prvkov na doske a dizajnu 3D tlačených súčiastok. Tieto zmeny sú podrobne v Kap. 3. Nové 3D tlačené súčiastky sú elegantnejšie a robustnejšie, tým sa stávajú odolnejšími voči fyzickému poškodeniu. Tiež kvôli zamedzeniu fyzického poškodenia bola vytvorená zarážka umiestnená vo vnútri trubičky na strane senzora, aby ho ochránila pred možnými nárazmi guličky. Rozloženie bolo zmenené tak, aby sa dalo pohodlne dočiahnuť na potenciometer aj počas behu experimentu.

Kapitola 4 je venovaná tvorbe a opisu programátorského rozhrania pre aplikácie v rôznych prostrediach. Prvým prostredím, je prirodzene Arduino IDE, nakoľko BoBShield je vytvorený tak, aby bol kompatibilný s doskami Arduino, s čím prichádza aj jednoduchosť používanie tohto prostredia. Keďže jazykom Arduino IDE je dialekt C/C++, pre vytvorenie programátorského rozhrania, resp. knižnice bolo nutné vytvoriť hlavičkový súbor. V tomto súbore je definovaná trieda objektov `BOBShield`, spolu s jej globálnymi premennými a metódami. Na ovládanie zariadenia BoBShield je potrebná jeho prvotná inicializácia. Netradične, knižnica pre BoBShield obsahuje dve inicializačné metódy, `BOBShield.begin()` a `BOBShield.initialize()`, čoho výhodou je umožnenie krátkeho testu, či je sensor zapojený správne v rámci inicializácie senzora. Metóda `BOBShield.calibration()` slúži na kalibráciu zariadenia. Kalibráciu nie je nutné vykonávať v každom programe, keďže minimálna a maximálna pozícia guličky sú prednastavené, kalibrovanie týchto hodnôt však umožní presnejšiu manipuláciu, predovšetkým v prípadoch, keď je poloha vyjadrená v percentách celkovej dosiahnuteľnej dĺžky tyče. Samotné odčítavanie polohy sa deje pomocou metóg `BOBShield.sensorRead()` a `BOBShield.sensorReadPerc()`, pričom výstupom prvej metódy je poloha vyjadrená v mm a výstupom druhej percento celkovej dosiahnuteľnej dĺžky. Manuálne nastavenie referenčnej hodnoty pomocou potenciometra se udeje s využitím metódy `BOBShield.referenceRead()`, kým metóda `BOBShield.actuatorWrite()` sa postará o natočenie motora o uhol, ktorý je vstupným argumentom tejto metódy. .

Podobná štruktúra bola implementovaná aj v prostredí Simulink. Príslušmé programové bloky Sensor block, Actuator block a Reference block boli vytvorené. Navyše sa vytvoril aj BoBShield block, zosobňujúci celé zariadenie. Úlohou Sensor blocku je čítanie nameraných hodnôt polohy zo senzora a ich prenesenie do Simulink schémy. Na toto bola využitá funkcionalita Simulinku S-Function, ktorá umožní užívateľovi použitie C/C++ knižnice od výrobcu v prostredí Simulink. Actuator block slúži na otočenie motora, pričom jej vstupom zo Simulink schémy je uhol, a výstupom samotné otočenie hardvérového prvku. Reference block je jednoduchý prepočet napäťových úrovní na pine zapojeného na potenciometer.

V Kap. 5 je odvodený model systému reprezentovaného zariadením, so vstupom do systému daným ako uhol natočenia podporného povrchu. Je dokázané, že z pohľadu identifikácie nie je zásadný rozdiel medzi modelom vychádzajúcim z translačného pohybu guličky a modelom, ktorý počíta s kotúľaním sa objektu. ďalší model, založený na diferenciálnej štvrtého rádu, ktorého vstupom do systému je moment, s očakávaním, že pre rozmery použitého hardvéru, sa tento dá vyjadriť ako súčin uhla natočenia a konštanty. V tejto kapitole je vykonaná aj

samotná identifikácia, počnúc návrhom správneho vstupného signálu pre zber dát. Tu sa naráža na problém, s ohraničeniami hardvéru, preto rôzne vstupné signály sú testované. V rámci identifikácie systému sa pristupuje k identifikácií prenosovej funkcie, lineárneho modelu s uhlom natočenia ako vstup, s rovnakým vstupom je uvedená identifikácia nelineárneho modelu a tiež modelu v tvare diferenciálnej rovnice štvrtého rádu. Výsledky sú v súlade s očakávaniami vyplývajúcimi z teórie: najmenšiu zhodu so skutočnými dátami má model v tvare prenosovej funkcie, kým medzi modelmi založenými na diferenciálnej rovnici druhého rádu (s uhlom natočenia ako vstupom) nie je signifikantný rozdiel v zhode s dátami, či už sa jedná o lineárny alebo nelineárny model, nelineárny model založený na diferenciálnej rovnici štvrtého rádu má najlepšiu zhodu celkovo, vďaka nahradeniu momentu násobkom uhla a konštanty však nedokáže dostatočne popísať zmeny systému v čase. Práve z uvedených dôvodov je pre didaktické príklady uvedené v Kap. 6 použitý model získaný identifikáciou lineárneho modelu z diferenciálnej rovnice druhého rádu.

Nasledujúca kapitola uvádza príklady použitia zariadenia BoBShield, a to príklad `Selftest`, jednoduchý test funkcionality prvkov na doske, `BOBShieldPID`, aplikáciu PID riadenia na reálny systém, `BOBShieldLQ`, aplikáciu LQ riadenia a na záver je ukázaný krátky príklad simulácie MPC riadenia systému, s porovnaním výsledkov pre rôzne dĺžky predikčného horizontu.

V Závere sú zhrnuté všetky výsledky tejto práce, s kritickým hodnotením a plánmi do budúcnosti, ako je napr. vývoj programátorského rozhrania pre aplikácie v iných prostrediach, implementovanie ďalších riadiacich algoritmov, pričom autor uvádza, že aj momentálna hardvérová konfigurácia v sebe ukrýva možnosti na ďalšie vylepšenie.

Appendix A

BOBShield: An Open-Source Miniature “Ball and Beam” Device for Control Engineering Educatioin

BOBShield: An Open-Source Miniature "Ball and Beam" Device for Control Engineering Education

Gergely Takács*, Erik Mikuláš, Anna Vargová, Tibor Konkoly, Patrik Šíma,
Lukáš Vadovič, Matúš Bíro, Marko Michal, Matej Šimovec and Martin Gulan
Institute of Automation, Measurement and Applied Informatics, Faculty of Mechanical Engineering
Slovak University of Technology in Bratislava, Bratislava, Slovakia
*gergely.takacs@stuba.sk

***Abstract*—This article presents a reference design for the well-known ball-on-beam laboratory experiment, where a spherical ball without direct actuation is only balanced by the inclination of a supporting structure, such as beam, rail or tube. The design introduced here is completely open-source and utilizes only a handful of off-the-shelf components and 3D printing; resulting in an exceptionally low hardware cost. Moreover, the resulting apparatus fits on a standard expansion module format, known as a Shield, which is compatible with a range of microcontroller prototyping boards from the Arduino ecosystem. This affordable, small, reproducible and open design is thus intended to aid control systems or mechatronics education via hands-on student experiments or even conducting research on a budget. In addition to the hardware design with downloadable project files, we also present an application programming interface and the results of a demonstration example here.**

***Index Terms*—open educational resources, control engineering education, microcontrollers, student experiments, mechatronics, educational technology**

I. Introduction

The ball and beam experimental device is a well-known staple of most laboratories teaching control engineering or mechatronics around the world. The apparatus consists of a surface with a dominant dimension, such as a beam, guiding rails or a tube that is actively inclined by an actuator. The goal of the feedback control system is to adjust the inclination such that a ball tracks a pre-set reference trajectory. This under-actuated, fast, nonlinear system is an excellent tool to test and teach concepts of control theory, modeling, identification, microcontroller technology and signal processing.

There is a range of vendors offering the ball and beam device in a commercial package: some examples include Quanser BB01 Ball and Beam, AMIRA ball and beam, Ball and Beam Control System Trainer Kit by ACROME, TecQuipment CE106 Ball and Beam Apparatus and others. Their commercial and standardized nature means that, in addition to education, they are suitable for research as well (c.f. [1], [2] and many others). Commercial ball and beam systems are high-quality precision products with excellent documentation, however, they usually take up extended bench space and are quite expensive and delicate.

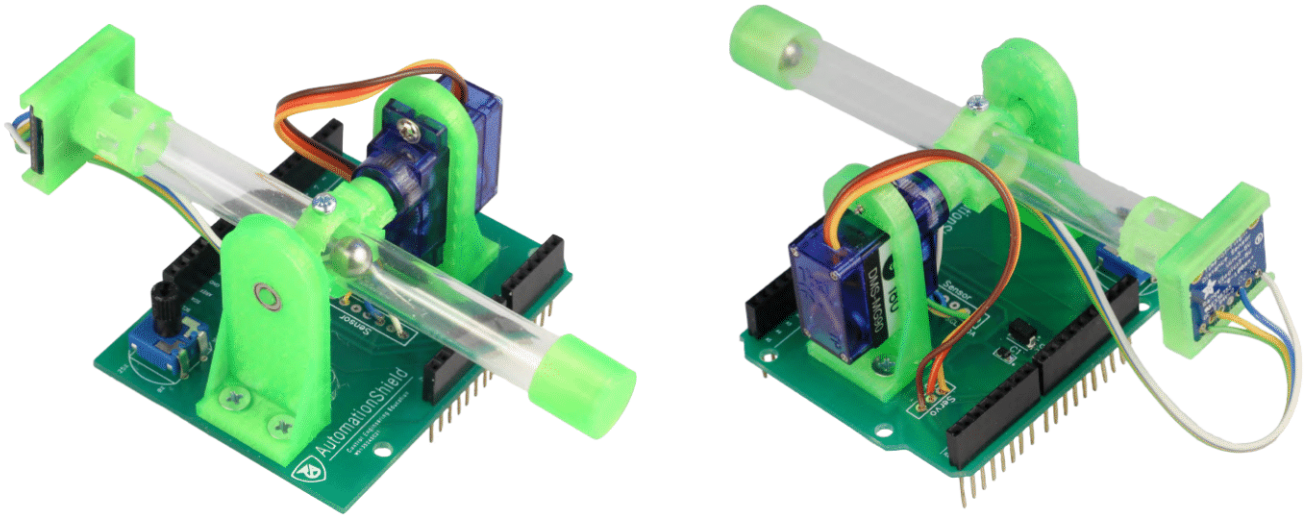
There is a plethora of research articles using ball and beam (also called ball on beam, ball-on-a-beam) systems for control algorithm development and verification, including neuro-fuzzy [1], model predictive [3], sliding mode [4], fractional order control [5] and others.

Numerous researchers choose an alternate route to equipping laboratories by designing and implementing their own version of the experiment (e.g. [6], [7]). In this case, the hardware is usually poorly documented, utilizes custom-made and improvised mechanical components and is specific to a course or research group. Hence, mainly due to their size and cost, neither of the aforementioned device categories is suitable for take-home student experiments or projects. We cannot fail to mention the importance of experimental laboratory devices that may be taken home in the ongoing COVID-19 pandemic caused by the SARS-CoV-2 virus [8] or aid online education and remote teaching at more ordinary times.

There are several examples of ball and beam devices in the literature (c.f. [9], [10]), some even use the popular and widely accepted Arduino microcontroller unit (MCU) prototyping platform instead of more expensive proprietary real-time computers or laboratory measurement cards. For example, Ahmad and Hussain introduce a relatively small and cheap ball on beam system based on the Arduino Uno, albeit using a rather improvised design [11]. Osinski et al. hint a more robust system using laser-cut mechanical parts in [7], however, do not offer a replicable documentation. An Arduino is employed by Kumar and Pandian as a measurement card without real-time control, utilizing yet again an improvised mechanical design [12]. Gao et al. have used Simulink to transcribe a PID block scheme to machine code for an Arduino Uno and a makeshift wooden ball and beam hardware [13]. There is even a ball on beam device design based on a cardboard construction available online [14]. A notable improvement in documentation quality and parts availability is presented in [15], where the Arduino-based ball and beam system is assembled from commercially available MakeBlock components.

In earlier work, we have proposed a concept of miniaturized, low cost, open-source laboratory devices that utilize the popular Arduino microcontroller prototyping platform for real-

The authors gratefully acknowledge the contribution of the Slovak Research and Development Agency (APVV) under the contracts APVV-18-0023, APVV-17-0214 and APVV-14-0399. The authors appreciate the financial support provided by the Cultural and Educational Grant Agency (KEGA) of the Ministry of Education of Slovak Republic under the contracts 005STU-4/2018 and 012STU-4/2021.



(a) Front view. (b) Back view.

Fig. 1. Isometric view of the assembled BOBShield device.

time feedback control. The electronic and mechanical layout known as the Arduino R3 pinout allows the design of hardware expansion modules, known in this ecosystem as "Shields". Formerly we have described a magnetic levitation apparatus [16], air flotation device [17], thermal experiment [18] and an optical system [19]. The key concept of our approach was low price, small footprint, universal and easy reproducibility.

Here we will introduce an open-source reference design of a low-cost miniaturized ball-on-beam (BOB) experimental device (Fig. 1). We will refer to this as the "BOBShield" for the remainder of the article.

II. Hardware

The following passages propose the reference design for the BOBShield device. The main aim of this section is to give as much information as possible, so that the device can be robustly replicated or even improved upon by others. Both mechanical and electronic components are marked alphabetically in the text and the reader may follow along the commentary by inspecting Figs. 2–5 and Table I, containing the same type of markings.

A. Mechanical

First, let us introduce the mechanical construction of the BOBShield. The entire device is built on a printed circuit board (PCB) copying the outline of the Arduino Uno (a). The two-layer FR4 board with 1.6 mm thickness and standard manufacturing tolerances fits into the customary 100 mm × 100 mm size limit, thus allows one to utilize the several small batch PCB manufacturers offering extremely competitive pricing. The BOBShield may be connected and mechanically fixed to any R3 pin layout microcontroller prototyping board by single row stacking header pins (b_1-b_3). The pair of 8- (b_1), 6- (b_2) and 10-pin (b_3) headers provide easy access to the

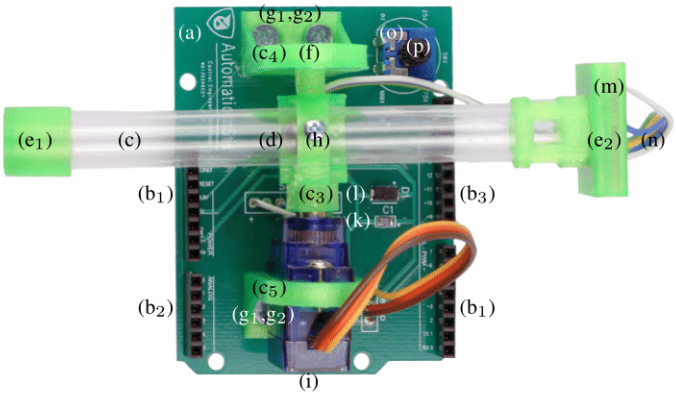


Fig. 2. Top view of the assembled device showing component marking.

electrical connections for external monitoring or data logging by e.g. an oscilloscope or a laboratory measurement card.

The 100 mm long transparent plexiglas tube (c) has an inner diameter of 10 mm and holds the 8 mm steel ball (d) that moves freely along its longitudinal axis, if inclined. One end of the tube is closed by a 3D printed cap (e_1), while the other end is terminated by another 3D printed part (e_2) designed to hold the distance measurement unit. The tube is fixed at its middle by a 3D printed circular clamp (e_3) that permits the inclination of the tube along its major axis. The clamp is held in place at both sides by L-shaped 3D printed brackets, one (e_4) containing a miniature ball bearing (f) to facilitate smooth movement and the other (e_5) fixed to the clamp through the shaft of the actuator. Figure 3 shows all of the 3D printed components (e_1–e_5), for which we provide downloadable CAD files [20]. We printed these components on an original Prusa i3 MK3/S 3D 3D printer using a total of 18 g filament under 3.5 h time. The L-shaped brackets are mounted to the PCB by M3 machine bolts (g_1) and fixed at the bottom by corresponding

nuts (g_2). The clamp is tightened to hold the tube steadily by a single self-cutting screw (h). The actuator is fixed to its mounting bracket by a pair of self-cutting screws that are bundled with the device, along with the usual horns which are not utilized in our design.

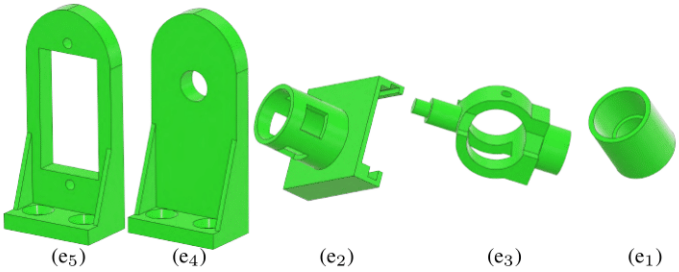


Fig. 3. 3D printed components.

B. Electronic

The electronic design of the BOBShield is kept as simple as possible, in order to reduce the complexity and price. The resulting electronic schematic drawing is shown in Fig. 4.

The inclination of the tube and, ultimately, the position of the ball, is manipulated by a standard micro RC Servo motor (i) with analog feedback. The total rotation range of the motor utilized in this application is only $\pm 30°$, thus any identically sized servo actuator will be suitable. The motor is connected to the `D9` pin of the standard Arduino R3 layout (j), since this equivalent MCU pin handles interrupts and timing on the Uno as well as other prototyping devices. The current consumption of the servo motor is low enough to be directly powered from the `5V` supply of the Arduino board without the need of an external wall-plug adapter. We added a capacitor (k) parallel to the motor to smooth out possible transients affecting the board supply and a diode (l) to prevent possible back-electromagnetic force damaging the circuitry.

The absolute distance of the ball from the reference located at one end of the tube is measured by the ST Microelectronics VL6180X time-of-flight (TOF) sensor (m). We found that this state-of-the art component compensates well for lightning conditions and surface types and we have not identified any significant problems with the reflective nature of the steel ball or the tube itself. Unfortunately, it comes in a physical package

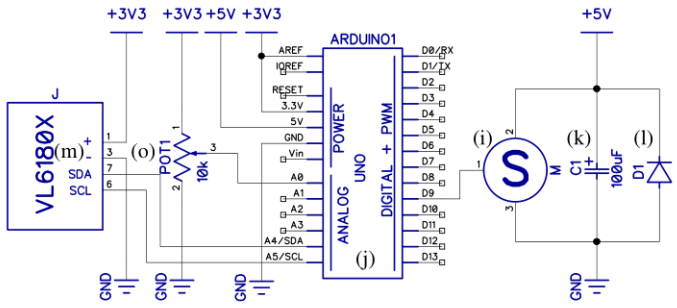


Fig. 4. Electronic schematics of the BoBShield device.

that is designed for industrial reflow soldering, therefore, its custom mounting is outside the reach of hobbyists and students. Hence, we have decided to utilize a widely available breakout board implementing the VL6180X for hobbyist use. This breakout may be purchased as an end-product and implements the sensor on a small-outline PCB with standard 0.1" pitch. The sensor communicates through the I2C protocol and we have utilized the standard `SCA` and `SCL` pins of the layout. The breakout connects to the base board via a flexible 4-wire ribbon cable (n). A flat flexible cable (FFC) would have been a better choice, unfortunately the sensor module would need to be custom-made for this.

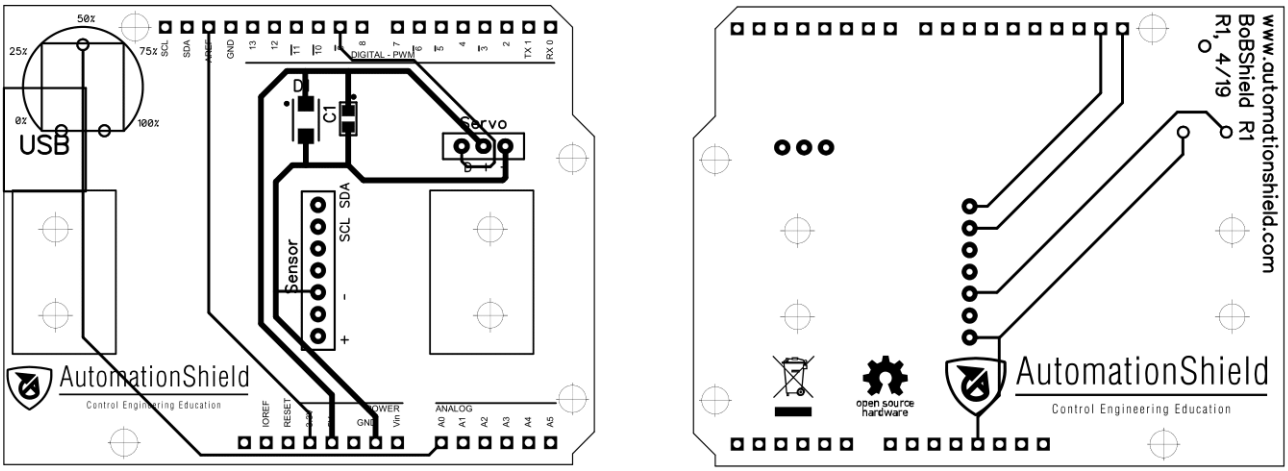
Finally, there is a potentiometer (o) with a small plastic shaft (p) that may be used for any user programmable role; but the main idea is to provide means to manually input the desired reference signal. The potentiometer is fed by the 3.3 V source for its maximal setting, so that the wiper connected to the `A0` analog port does not damage other microcontroller boards—such as the Due or Zero—operating with CMOS logic levels. To maximize analog resolution, we have also connected the `3.3V` source to the `AREF` pin and compensate for this change in code. The sensor board is also powered by the 3.3 V source.

The resulting two-layer printed circuit board is shown in Fig. 5. We have avoided including further header pins and connectors and soldered the wires connecting to the servo and sensor directly to the PCB. The electronic schematics and the PCB were designed in the free version of DIP Trace, making further changes accessible to anyone. We provide the editable schematic files and the editable PCB layout, along with the manufacturing-ready Gerber format PCB [20].

C. Cost and part availability

A comprehensive parts list with a detailed description of our component choices and pricing is listed in Tab. I. The pricing excludes postage, consumables (solder, flux, etc.) and labor cost. Unit prices have been determined by searching for the cheapest possible alternatives using meta-search engines, through a range of various vendors and for relatively small quantity purchases that have been for <100 pieces in all cases. We have also included auction sites, since these may offer lower-priced small quantity purchases than established electronic component vendors. We believe that this calculation can be realistic even for a larger-scale production, since the unaccounted for items are balanced by deals on parts. The total price will approximately double, if parts are sourced from a single supplier in low quantities and time is an issue. We have decided not to include the price of the Arduino board in our figure, as these devices are useful in other roles as well and the BOBShield is compatible with several alternatives.

The roughly $10 U.S. final material cost makes our proposed design affordable to any institution teaching mechatronics or control engineering concepts, even in low-income countries. The device is well within the reach of individual students too; making it easy to distribute for projects, assignments or for remote teaching.



(a) Front side.

(b) Back side.

Fig. 5. The printed circuit board of the BoBShield device.

We have taken care to only include parts that are easily available and can be globally sourced. Due to its open-source nature, the sensor board is offered by various vendors. Practically any size-compatible servo motor will work, and the rest of the components can be effortlessly exchanged to equivalent alternatives.

III. Application Programming Interface

This section introduces the application programming interface for the BOBShield device, written in C/C++ for the Arduino IDE. The goal of the API is to abstract basic hardware functionality to input-output functions, thus encouraging the users to quickly create their own control applications. It goes without saying, that letting students to write their own interface holds educational value as well.

The API is contained within the `BOBShield.h` header file of the open-source AutomationShield library [21]. Since every implementation file (`*.cpp`) is compiled in a given Arduino library regardless of actual usage, the servo motor and sampling subsystems would conflict in re-assigning interrupt service vectors. Thus we have sacrificed good programming practice for the sake of didactic value, since the AutomationShield library must handle the API and application examples for several other devices.

The hardware functionality is handled by the `BOBClass` class which creates a `BOBShield` object. As per Arduino custom, the initialization functionality is contained in the `begin()` method and calling

```
BOBShield.begin();
```

will start the servo motor functionality by assigning its hardware pin for later use. The sensor readings can be calibrated by using the

```
BOBShield.calibration();
```

method, which tilts the beam towards $\pm 30^\circ$ and selects the maximal, respectively, the minimal distance readings. These calibrated readings are accessible to other methods.

All I/O functionality is handled by three methods, that have been named `read()` and `write()` according to Arduino nomenclature. Using

```
y = BOBShield.sensorRead();
```

will output the absolute position of the ball y (mm) as a floating-point number by polling the ToF sensor. Similarly, the percentual position based on calibration may be accessed by the `sensorReadPerc()` method. The manual reference from the potentiometer is read by the

```
r = BOBShield.referenceRead();
```

method, returning the reference r as a floating-point number. Finally, the desired angle u (°) is sent to the servo motor by

```
BOBShield.actuatorWrite(u);
```

where the input parameter is a floating point number.

The rest of the AutomationShield library for the Arduino IDE provides numerous universal functions that are common to feedback control, such as, a comprehensive interrupt-based sampling framework, a PID routine, signal processing, Kalman filtering, etc. The library, including the BOBShield API, is compatible with microcontroller architectures other than the Atmel AVR assumed for the widely used Arduino Uno/Mega2560. For example, the code shall be compatible with various other devices with ARM Cortex M architecture MCU, e.g. Arduino Due (Atmel SAM3X8E), Arduino Zero (Atmel SAMD21), Adafruit Metro M4 Express (Microchip ATSAMD51), etc.

IV. Examples

In addition to the API, the AutomationShield library contains demonstration examples for each device. Currently, there are four examples offered for the BOBShield Arduino API.

TABLE I
COMPONENT LIST AND COST CALCULATION FOR MATERIAL PRICE IN U.S. DOLLARS.

Name	Part no., value.	PCB	Mark	Pcs.	Unit	Total
PCB	2-layer, FR4, 1.6 mm thick (e.g. JCLPCB)	-	(a)	1	0.40	0.40
3D print	18 g, $\phi = 1.75$ mm PETG filament, bright green, at 240 °C (90 °C bed), 3 h 20 min	-	(e_{1-5})	1	0.64	0.64
Trimmer	10 kΩ, 250 mW, single turn THT trimmer (e.g. ACP CA14NV12,5-10KA2020)	POT1	(o)	1	0.19	0.19
Screw	DIN 7971C 2.2 × 6.5	-	(h)	1	0.02	0.02
Diode	generic diode, DO214AC (e.g. Vishay Semiconductor BYG20J)	D1	(l)	1	0.38	0,38
Screws	M3×8 DIN963A	-	(g_1)	4	0.02	0.08
Nuts	M3 DIN439B	-	(g_2)	4	0.01	0.04
Header	10×1, female, stackable, 0.1“ pitch (e.g. SparkFun 474-PRT-10007)	-	(b_3)	1	0.09	0.09
Pot shaft	For “POT1”, (e.g. ACP CA9MA9005)	-	(p)	1	0.07	0.07
Header	8×1, female, stackable, 0.1“ pitch (e.g. SparkFun 474-PRT-10007)	-	(b_1)	2	0.06	0.12
Header	6×1, female, stackable, 0.1“ pitch (e.g. SparkFun 474-PRT-10007)	-	(b_2)	1	0.06	0.06
Tube	transparent, plexiglass (PMMA), L=100 mm, 12/10 mm	-	(c)	1	0.38	0.38
Sensor	VL53L1X breakout, time of flight ranging SNSR	J	(m)	1	4.20	4.20
Motor	Metal geared 9 g, 5 V micro servo	M	(i)	1	1.92	1.92
Capacitor	100 uF, 6.3V, 20, 0805, tantalum	C1	(k)	1	0.53	0.53
Ball	Steel ball, diameter 8 mm	-	(d)	1	0.18	0.18
Bearing	Miniature ball bearing, 3×6×2 mm, (e.g. MR-63-ZZ)	-	(f)	1	0.26	0.26
Cable	Ribbon cable, 4-7 wires, cca. 0.2 m (e.g. EL-2468)	-	(n)	1	0.08	0.08
					Total:	**$9.46**

The file `BOBShield_SelfTest.ino` implements a basic self-test routine, aiding the verification of the hardware functionality. First, the ToF sensor is initialized and, if the MCU cannot connect to the sensor chip, will report the failure. Then, the servo turns to each of its extreme operation points and the routine compares the distance readings with expected values.

The project file `BOBShield_Identification.ino` performs an open-loop test by supplying a range of actuator settings, while `BOBShield_Identification_aprbs.ino` does the same but applies an amplitude-modulated pseudo-random signal (Fig. 6). Both examples assume a strict interrupt-based sampling framework and print the input u and corresponding output y to the serial line. This feed can be recorded with an arbitrary serial terminal software and later used as input-output data for system identification and mathematical modelling.

Finally, `BOBShield_PID.ino` demonstrates the proportional-integral-derivative (PID) position control of the steel ball. Besides the engaging visual action of the hardware, results are also printed to the serial line. Figure 7 demonstrates a step change of the reference, where the process variables r (blue), y (red) and u (green) may be followed in the oscilloscope-like rolling window of the Arduino IDE Serial Plotter. Similar to the open-loop case, any other terminal program is suitable for monitoring and data logging.

The PID demonstration example is sampled with a $T = 10$ ms period, and a pre-set reference vector of $[30, 50, 8, 65, 15, 40]^{\mathrm{T}}$ mm is requested for 1000 samples (10 s) providing sufficient time for transients. The PID controller is manually tuned to the $K_{\mathrm{P}} = 0.3$ (mm/°), $T_{\mathrm{I}} = 600$ and $T_{\mathrm{D}} = 0.22$ constants. An integral windup clipping strategy is set to ±10 (°), while the input to the servo motor is saturated at $u = 30$ (°). The results of this experiment are illustrated in Fig. 8, where reference position r is compared to achieved

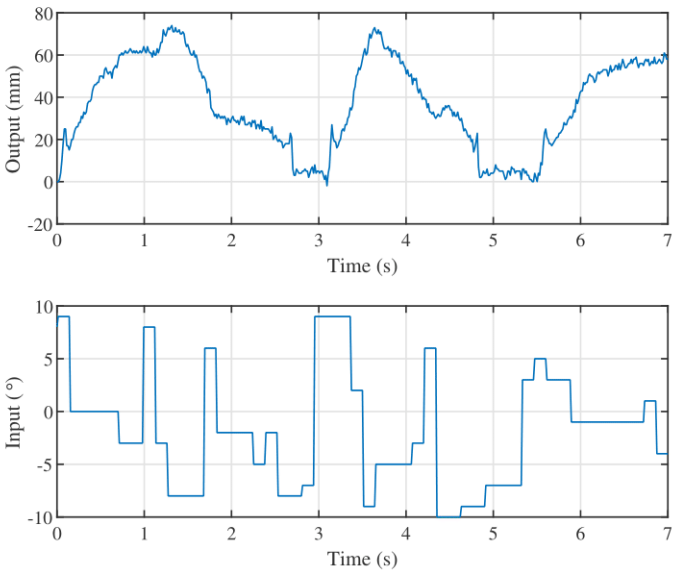


Fig. 6. Open-loop response for system identification.

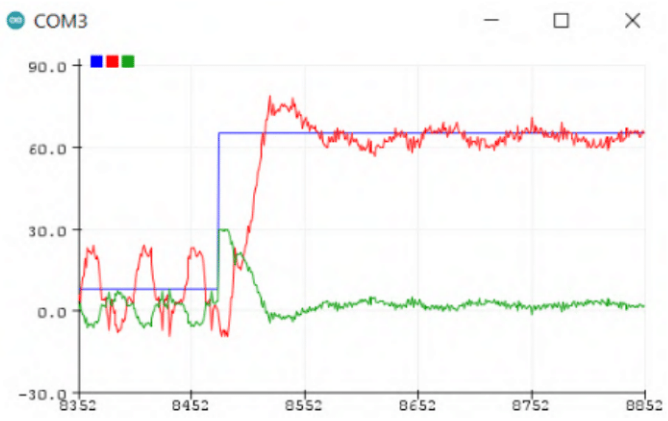


Fig. 7. Screenshot of a PID experiment in the Arduino IDE Serial Plotter.

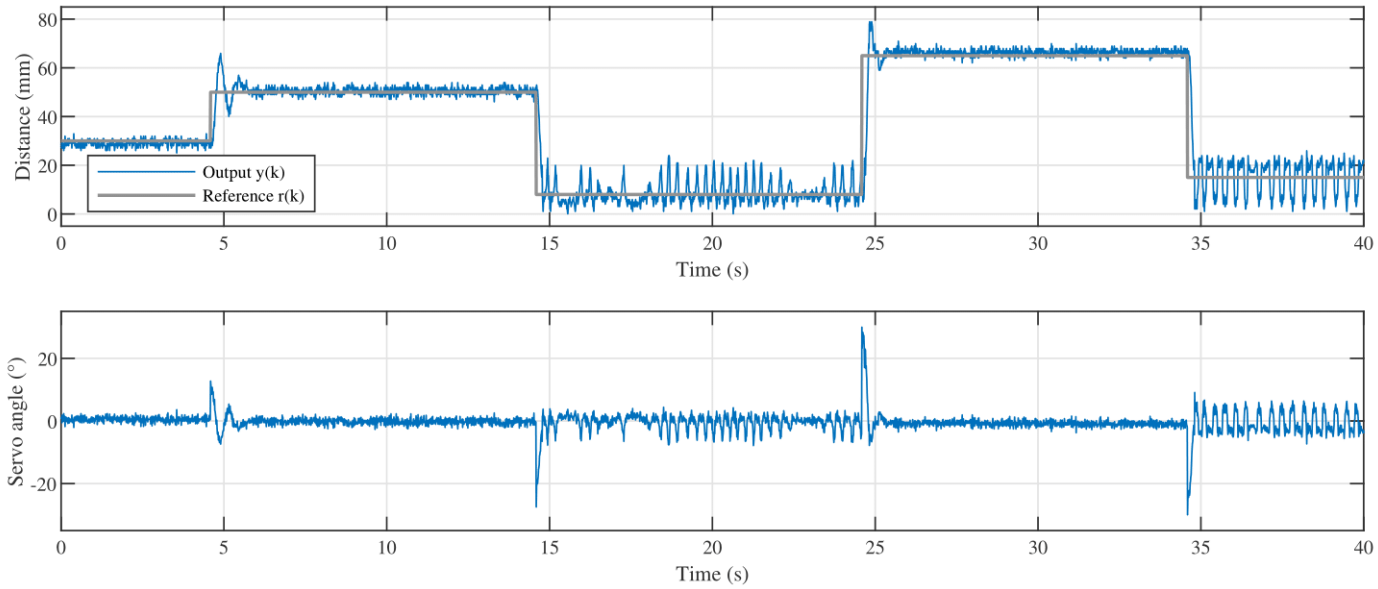


Fig. 8. PID control of the ball position.

output y on the top and the corresponding servo input u is presented at the bottom.

The ball follows the reference position trajectory faithfully across a wide span of the working range. The tracking is worse when the ball is close to the sensor, we believe this is caused by sensor noise originating from unwanted reflections or other physical effects. Since the main point of this article is to present our hardware design, and due to the limited scope we will not introduce other control methods here. However, the reader shall note that it is likely that the response shown in Fig. 8 may be improved by more advanced control and/or appropriate signal processing.

V. Conclusion

This paper presented a low-cost, open-source "ball-on-beam" didactic device that can be physically attached to a range of Arduino-compatible microcontroller prototyping boards and is suitable for control and mechatronics education. In addition to the hardware, we introduced the input-output interface and demonstrative examples.

Implementing this reference design results in an apparatus with a material cost below $10 and a small physical outline, making it suitable for classroom distribution and take-home student experiments. The open hardware, interface and examples foreshadow the possibility of collaboration between various educators and instructors in bettering the API, examples and even creating open teaching materials.

Future work shall improve the hardware design mainly by making the 3D printed parts more robust and resilient. We are also planning to expand the current API to MATLAB, Simulink, Python and possibly a range of other scientific and engineering software, such as LabVIEW, Octave and Scilab/Xcos. The range of demonstration examples shall be broadened by the addition of other control engineering methods, such as, linear quadratic (LQ) or model predictive control (MPC).

Acknowledgements

The authors would like to thank the involuntary assistance of Marek Krippel and Rastislav Haška with the original version of the documentation. We are still not really sure what Samuel Mladý was working on, but thanks for sticking around.

References

[1] N. Eqra, R. Vatankhah, and M. Eghtesad, "A novel adaptive multi-critic based separated-states neuro-fuzzy controller: Architecture and application to chaos control," *ISA Transactions*, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0019057820304997

[2] I. M. Mehedi, U. M. Al-Saggaf, R. Mansouri, and M. Bettayeb, "Two degrees of freedom fractional controller design: Application to the ball and beam system," *Measurement*, vol. 135, pp. 13–22, 03 2019.

[3] N. Hara, M. Takahashi, and K. Konishi, "Experimental evaluation of model predictive control of ball and beam systems," in *Proceedings of the 2011 American Control Conference*, 2011, pp. 1130–1132.

[4] S. Sameera, R.; Arun, "Position tracking of ball and beam system using fractional order controller," *International Journal of Advanced Research Trends in Engineering and Technology*, vol. 4, no. 6, pp. 174–180, 2017.

[5] T. Emami, "Mixed sensitivity design of pid controller-applied to a ball and beam system," in *ASME 2017 International Mechanical Engineering Congress and Exposition*, vol. 4. ASME, 2017.

[6] M. Saad and M. Khalfallah, "Design and implementation of an embedded ball-beam controller using PID algorithm," *Universal Journal of Control and Automation*, vol. 4, pp. 63–70, 2017.

[7] C. Osinski, A. L. R. Silveira, C. Stiegelmaier, M. G. Bergamini, and G. V. Leandro, "Control of ball and beam system using fuzzy PID controller," in *2018 13th IEEE International Conference on Industry Applications (INDUSCON)*, 2018, pp. 875–880.

[8] K. Bangert, J. Bates, S. Beck, Z. Bishop, M. D. Benedetti, J. Fullwood, A. Funnell, A. Garrard, S. Hayes, T. Howard, C. Johnson, M. Jones, P. Lazari, J. Mukherjee, C. Omar, B. Taylor, R. Thorley, G. Williams, and R. Woolley, "Remote practicals in the time of coronavirus, a multi-disciplinary approach," *International Journal of Mechanical Engineering Education*, vol. 0, no. 0, p. 0306419020958100, 0.

[9] M. M. Kopichev, A. A. Kuznetsov, A. R. Muzalevskiy, and T. L. Rusyaeva, "Ball and beam stabilization laboratory test bench with intellectual control," in *2020 XXIII International Conference on Soft Computing and Measurements (SCM)*, 2020, pp. 112–116.

[10] G. Dewantoro, D. Susilo, and D. C. Amanda, "Development of microcontroller-based ball and beam trainer kit," *Indonesian Journal of Electrical Engineering and Informatics*, pp. 45–54, 03 2015.

[11] B. Ahmad and I. Hussain, "Design and hardware implementation of ball beam setup," in *2017 Fifth International Conference on Aerospace Science Engineering (ICASE)*, 2017, pp. 1–6.

[12] S. T. Kumar and B. J. Pandian, "Tuning and implementation of fuzzy logic controller using simulated annealing in a nonlinear real-time system," in *2014 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2014]*, 2014, pp. 916–921.

[13] Z. Gao, S. Wijesinghe, T. Pathinathanpillai, E. Dyer, and I. Singh, "Design and implementation a ball balancing system for control theory course," 2015.

[14] K. Benchikha, "Ball and Beam: PID control on Arduino with LabView to stabilize a ball on a beam," Online, 2019, [cited 11.27.2020]; Available from https://create.arduino.cc/projecthub/karem_benchikha/ball-and-beam-601d7a. Arduino Project Hub.

[15] Wolfram Research, "Microcontroller Kit Sample Projects: Balancing a ball on a beam," Online, 2020, [cited 11.27.2020]; Available from https://reference.wolfram.com/language/MicrocontrollerKit/workflow/BallAndBeamControl.

[16] G. Takács, J. Mihalík, E. Mikuláš, and M. Gulan, "MagnetoShield: Prototype of a low-cost magnetic levitation device for control education," in *Proceedings of the 2020 IEEE Global Engineering Education Conference (EDUCON)*, Porto, Portugal, April 2020, pp. 1516–1525.

[17] G. Takács, P. Chmurčiak, M. Gulan, E. Mikuláš, J. Kulhánek, G. Penzinger, M. Podbielančík, M. Lučan, P. Šálka, and D. Šroba, "FloatShield: An open source air levitation device for control engineering educationn," in *Proceedings of the 2020 IFAC World Congress*, Berlin, Germany, July 2020, pp. 1–8, (Preprints).

[18] G. Takács, M. Gulan, J. Bavlna, R. Köplinger, M. Kováč, E. Mikuláš, S. Zarghoon, and R. Salíni, "HeatShield: a low-cost didactic device for control education simulating 3D printer heater blocks," in *Proceedings of the 2019 IEEE Global Engineering Education Conference*, Dubai, United Arab Emirates, April 2019, pp. 374–383.

[19] G. Takács, T. Konkoly, and M. Gulan, "Optoshield: A low-cost tool for control and mechatronics education," in *Proceedings of the 12th Asian Control Conference*, Kitakyushu-shi, Japan, Jun 2019, pp. 1001–1006.

[20] G. Takács *et al*., "BOBShield," Online, 2020, [cited 1.12.2020]; GitHub Wiki page documenting the BOBShield device. Available from https://github.com/gergelytakacs/AutomationShield/wiki/BoBShield.

[21] ——, "AutomationShield: Control Systems Engineering Education," Online, 2020, [cited 11.24.2020]; Available from http://www.automationshield.com. 2018–2020.

Appendix B

```
startScript;                                % Clears screen
   and variables, except allows CI testing
%% Choose the type of grey-box model to identify
model = 'nonlinear';                                %
   Nonlinear state-space model
%model = 'linearSS';                                 % Linear
    state-space model
%model = 'linearTF';                                 % Linear
    transfer function
%% Choose the ODE for nl grey-box model
%ODE ='linearODE';   %only for check
ODE ='sinus';
%ODE ='sinus2';
%ODE ='4thorder';
%% Data preprocessing
%load data from experiment
load('dataAprbsAllNEW.mat')
u=dataAll(:,1)*pi/180; %input in rad
y=dataAll(:,2)/1000; %output in m
%moving average filtering of y
% windowSize = 5;
% b = (1/windowSize)*ones(1,windowSize);
% a = 1;
% y = filter(b,a,y);
 Ts = 0.01;                                         %
    Sampling
data = iddata(y,u,Ts,'Name','Experiment');          % Create
   identification data object
% subplot(211); plot(data)
 data = data(4300:4450);                             % Select
     a relatively stable segment
% data = data(5600:5700);
% subplot(212); plot(data)
% return
data = detrend(data);                               % Remove
   offset and linear trends
```

```
data.InputName = 'Servo Angle';                    % Input
    name
data.InputUnit =  'deg';                           % Input
    unit
data.OutputName = 'Ball Position';                 % Output
   name
data.OutputUnit = 'm';                             % Output
   unit
data.Tstart = 0;                                   %
   Starting time
data.TimeUnit = 's';                               % Time
   unit

%% Initial guess of model parameters
%initial estimation of parameter H
m= 2.101e-3; % mass of the ball in kg
g=9.81; %gravitational acceleration
J = 9.99e-6; % ball's moment of inertia
R=0.0037; %radius of the ball in m
H = m*g;
bb=1;
L=0.04; %m
h=(H*2*R)/L;
M=1e-3; %mass of the beam

r0  = data.y(1,1);                                 % Initial
    ball position estimate
dr0 = (data.y(2,1)-data.y(1,1))/Ts;                % Initial
    ball velocity estimate
alpha0  = 30;                                       %
   Initial motor position estimate (after calibration)
dalpha0=0;                                         %
   Initial motor velocity estimate
%%

switch model
    case 'nonlinear'
 switch ODE
    case 'linearODE' %only for check
           % Model structure
    FileName     = 'BOBShield_ODE';                % File
       describing the model structure
    Order          = [1 1 2];                       %
       Model orders [ny nu nx]
    Parameters     = [H];            % Initial values of
       parameters
```

```
    InitialStates = [r0;dr0];                          %
       Initial values of states
    Ts            = 0;                                 %
       Time-continuous system
%%
     % Set identification options
     nlgr = idnlgrey(FileName,Order,Parameters,
        InitialStates,Ts,'Name','Nonlinear Grey-box Model
        ');
     set(nlgr,'InputName','Servo Angle','InputUnit', 'deg
        ','OutputName','Ball Position','OutputUnit','m','
        TimeUnit','s');
     %nlgr = setpar(nlgr,'Name',{'Supercalifragilistic
        Coefficient'});
     nlgr = setinit(nlgr,'Name',{'Ball Position' 'Ball
        Speed'});
     nlgr.InitialStates(1).Fixed = false;
     nlgr.InitialStates(2).Fixed = false;
     nlgr.Parameters(1).Fixed = false;
%      nlgr.Parameters(2).Fixed = false;
%      nlgr.Parameters(3).Fixed = false;
%      nlgr.Parameters(4).Fixed = false;

     size(nlgr);
    case 'sinus'
           % Model structure
    FileName     = 'BOBShieldSin_ODE';                 %
       File describing the model structure
    Order        = [1 1 2];                            %
       Model orders [ny nu nx]
 %   Parameters    = [J,R,m,M,g];            % Initial
    values of parameters
 %   Parameters    = [H];             % Initial values of
    parameters
    Parameters    = [H];            % Initial values of
       parameters

    InitialStates = [r0;dr0];                          %
       Initial values of states
    Ts            = 0;                                 %
       Time-continuous system
%%
     % Set identification options
     nlgr = idnlgrey(FileName,Order,Parameters,
        InitialStates,Ts,'Name','Nonlinear Grey-box Model
```

```
        ');
     set(nlgr,'InputName','Servo Angle','InputUnit', 'deg
        ','OutputName','Ball Position','OutputUnit','m','
        TimeUnit','s');
     %nlgr = setpar(nlgr,'Name',{'Supercalifragilistic
        Coefficient'});
     nlgr = setinit(nlgr,'Name',{'Ball Position' 'Ball
        Speed'});
     nlgr.InitialStates(1).Fixed = false;
     nlgr.InitialStates(2).Fixed = false;
     nlgr.Parameters(1).Fixed = false;
%      nlgr.Parameters(2).Fixed = false;
%      nlgr.Parameters(3).Fixed = false;
%      nlgr.Parameters(4).Fixed = false;

     size(nlgr);
    case 'sinus2'
           % Model structure
    FileName     = 'BOBShieldSin2_ODE';              %
       File describing the model structure
    Order          = [1 1 2];                        %
       Model orders [ny nu nx]
    Parameters     = [J,R,m,M,g];            % Initial
       values of parameters
 %   Parameters     = [H];            % Initial values of
    parameters
 %   Parameters     = [H];            % Initial values of
    parameters

    InitialStates = [r0;dr0];                       %
       Initial values of states
    Ts             = 0;                             %
       Time-continuous system
%%
     % Set identification options
     nlgr = idnlgrey(FileName,Order,Parameters,
        InitialStates,Ts,'Name','Nonlinear Grey-box Model
        ');
     set(nlgr,'InputName','Servo Angle','InputUnit', 'deg
        ','OutputName','Ball Position','OutputUnit','m','
        TimeUnit','s');
     %nlgr = setpar(nlgr,'Name',{'Supercalifragilistic
        Coefficient'});
     nlgr = setinit(nlgr,'Name',{'Ball Position' 'Ball
        Speed'});
     nlgr.InitialStates(1).Fixed = false;
```

```
     nlgr.InitialStates(2).Fixed = false;
     nlgr.Parameters(1).Fixed = false;
%      nlgr.Parameters(2).Fixed = false;
%      nlgr.Parameters(3).Fixed = false;
%      nlgr.Parameters(4).Fixed = false;

     size(nlgr);
    case '4thorder'   % Model structure
    FileName     = 'BOBShield4th_ODE';              %
       File describing the model structure
    Order          = [1 1 4];                       %
       Model orders [ny nu nx]
 %   Parameters     = [J,R,m,M,g];            % Initial
    values of parameters
 %   Parameters     = [H];            % Initial values of
    parameters
    Parameters     = [H, bb, h];            % Initial
       values of parameters

    InitialStates = [r0;dr0;alpha0;dalpha0];
                          % Initial values of states
    Ts             = 0;                             %
       Time-continuous system
%%
     % Set identification options
     nlgr = idnlgrey(FileName,Order,Parameters,
        InitialStates,Ts,'Name','Nonlinear Grey-box Model
        ');
     set(nlgr,'InputName','Servo Angle','InputUnit', 'deg
        ','OutputName','Ball Position','OutputUnit','m','
        TimeUnit','s');
     %nlgr = setpar(nlgr,'Name',{'Supercalifragilistic
        Coefficient'});
     nlgr = setinit(nlgr,'Name',{'Ball Position' 'Ball
        Speed' 'Angle' 'Angular speed'});
     nlgr.InitialStates(1).Fixed = false;
     nlgr.InitialStates(2).Fixed = false;
     nlgr.Parameters(1).Fixed = false;
%      nlgr.Parameters(2).Fixed = false;
%      nlgr.Parameters(3).Fixed = false;
%      nlgr.Parameters(4).Fixed = false;

     size(nlgr);
end
 %%
     % Identify model
```

```
      opt = nlgreyestOptions('Display','on','EstCovar',
         true,'SearchMethod','Auto'); %gna
      opt.SearchOption.MaxIter = 50;                  %
         Maximal number of iterations
      model = nlgreyest(data,nlgr,opt);               % Run
          identification procedure

      case 'linearSS'
      order=4;
      switch order
          case 2
     % Model structure
%initial estimation of the system
A = [0 1 ; 0 0];
B = [0 H]';
C = [1 0];
D = [0];

     K = zeros(2,1);                                  %
        Disturbance
     K(1) = 5;                                       % State
        noise

     x0 = [r0; dr0];                                % Initial
        condition
     disp('Initial guess:')
     sys = idss(A,B,C,D,K,x0,0)                       %
        Construct state-space representation

     % Mark the free parameters
     sys.Structure.A.Free = false;
     sys.Structure.B.Free = [0  1]';                 % Free and
        fixed variables
     sys.Structure.C.Free = false;                    % No free
        parameters

      sys.DisturbanceModel = 'none';                 % Estimate
         disturbance model
     sys.InitialState = 'estimate';                   % Estimate
         initial states

     % Set identification options
      Options = greyestOptions;                        % State
         -space estimation options
     Options = ssestOptions;                          % State-
        space estimation options
```

```
    Options.Display = 'on';                          % Show
       progress
    Options.Focus = 'simulation';                    % Focus on
        simulation
                                                     % '
                                                        stability
                                                        '
    Options.InitialState = 'estimate';               % Estimate
        initial condition
    % Identify model
    model = ssest(data,sys,Options);                 % Run
       estimation procedure

         case 4
    % Model structure
%initial estimation of the system
A = [0 1 0 0 ; 0 0 H 0; 0 0 0 1; 0 0 0 0];
B = [0 0 0 1]';
C = [1 0 0 0];
D = [0];

    K = zeros(4,1);                                  %
       Disturbance
    K(1) = 5;                                       % State
       noise

    x0 = [r0; dr0;0;0];                              % Initial
       condition
    disp('Initial guess:')
    sys = idss(A,B,C,D,K,x0,0)                       %
       Construct state-space representation

    % Mark the free parameters
    sys.Structure.A.Free = [0 0 0 0 ; 0 0 1 0; 0 0 0 0; 0
        0 0 0];
    sys.Structure.B.Free = false;                % Free and
       fixed variables
    sys.Structure.C.Free = false;                    % No free
       parameters

     sys.DisturbanceModel = 'none';                % Estimate
        disturbance model
    sys.InitialState = 'estimate';                   % Estimate
        initial states

    % Set identification options
```

```
    Options = greyestOptions;                    % State
       -space estimation options
    Options = ssestOptions;                      % State-
       space estimation options
    Options.Display = 'on';                      % Show
       progress
    Options.Focus = 'simulation';                % Focus on
        simulation
                                                 % '
                                                    stability
                                                    '
    Options.InitialState = 'estimate';           % Estimate
        initial condition
    % Identify model
    model = ssest(data,sys,Options);             % Run
       estimation procedure
    end

    case 'linearTF'

    % Model structure
    model = idproc('P2I');                      % Second order
       with integrator

    model.Structure.Kp.Value=10;                 % Initial gain
    model.Structure.Kp.Maximum=10;              % Maximal gain
%    model.Structure.Kp.Minimum=5;               % Minimal gain

    model.Structure.Tp1.Value=0.2;              % Initial time
       constant
%     model.Structure.Tp1.Maximum=0.5;           % Maximal time
   constant
    model.Structure.Tp1.Minimum=0;              % Minimal time
       constant

    model.Structure.Tp2.Value=0.5;             % Initial time
       constant
%     model.Structure.Tp2.Maximum=0.9;           % Maximal time
   constant
    model.Structure.Tp2.Minimum=0;              % Minimal time
       constant

    % Set identification options
    Opt = procestOptions;                       % Estimation
       options
```

```
    Opt.InitialCondition = 'estimate';  % Estimate the
       initial condition
    Opt.DisturbanceModel = 'ARMA2';     % Define noise
       model
    Opt.Focus = 'stability';            % Goal is to
       create a stable model
    Opt.Display = 'full';               % Full estimation
        output

    % Identify model
    model = procest(data,model,Opt);    % Estimate a
       process model

end

compare(data,model);                            % Compare
   data to model
model                                           % List
   model parameters
grid on                                         % Turn on
   grid

return
```